Snowflakes
29457A

29457A

Engineering Notebook
High School

HIGH STAKES 24-25

St Chris Snowflakes

Table of Contents

Entries

Table of Contents

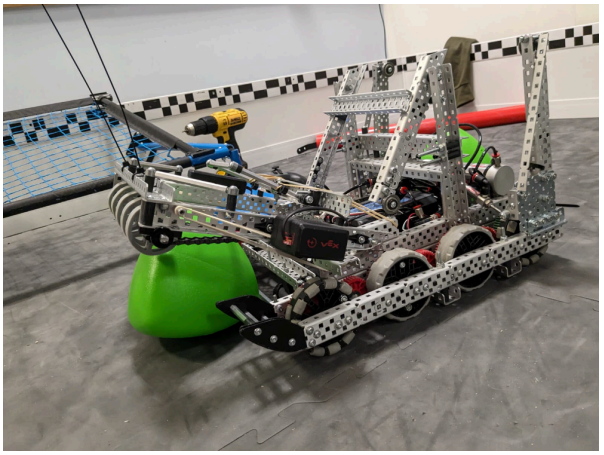
Appendix

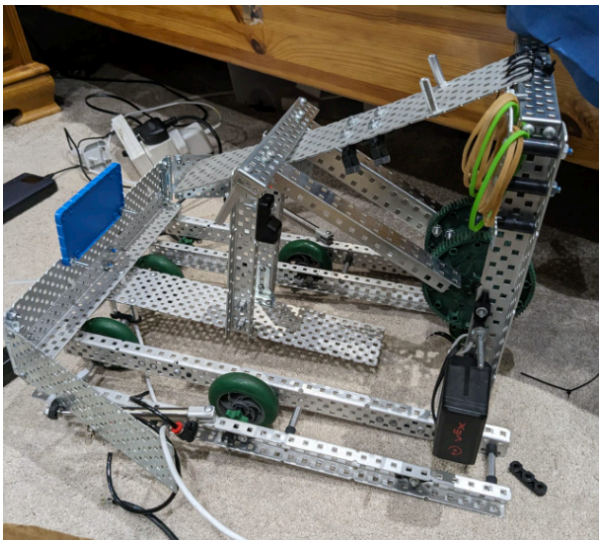
Team Introduction

Who are the Snowflakes?

We are the St Chris Snowflakes; we started out as a VRC team in September 2023, in our first season as a team in 'Over Under'. We set out to do our very best and we quickly found that we all loved the challenge of VEX and wanted to excel as far as we could; after a struggle, we qualified for UK nationals and siezed the oppurtunity to become the best we could. Our hard work payed off, and we went home with design award and a spot in the VEX World Championships; where we went in April and gained key experience to start this season.

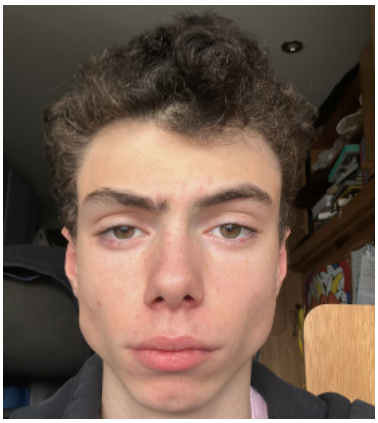
Off of the back of Over Under we have acquired some very important skills necessary for competitive representation on both a national and global scale such as good building and coding practices which we hope to utilise in order to become even more competitive in all aspects of the V5RC competition where we weren't last year. We are also moving to a smaller team size as some of our members are moving on. This will mean we may have to work harder but also means a larger range of experience in other areas.





Team Introduction

The members



Daniel Dew
- Captain
- CAD Designer
- Head Programmer
- Driver

My role within the team primarily resides in the programming of the robot's code including autonomous and driver control. I also aid with the design and CAD phases of building. I am also the primary driver.

Daniel da Silva
- Programmer
- Primary Notebooker

Hi I'm Daniel da Silva and my role within the team is to support the development of the robot's software and to help organise it into the logbook.

Jonah Fitchew
- Co-Head Builder
- Reserve Driveteam

Hello I am Jonah Fitchew and my role within the team is the physical construction of the robot, documentation and assisting the driver during matches.

Aubert Seysess
- Co-Head Builder
- CAD Designer
- Driveteam
- Head Chef

Hello I am Aubert Seysses and my role within the team is to help design and virtually CAD the

robot, also, I aid with the physical construction of the robot.

Thomas Robb

- Head Tactics
- Driveteam
- Mentor

My role within the team is to brainstorm tactics and communicate with the other members to ensure that designs and tactics align. During competitions I am also responsible for taking note of performances, both our's and other team's; to help find possible alliances.

How To Use This Notebook

About this Notebook

TL;DR

For this season, we decided to deviate from the standard process for making engineering notebooks. We decided that, with the loss of our main logbooker we would have to share the notebooking duties; this meant that formatting could become inconsistent and we immediatly found that it took too long to format everything to the desired (exceptional) standard. Therefore, to cut down on time and improve the notebook's readability and functionality, we decided to adopt the *Notebookinator* template, which is an extension of the *typst* markup language.

Why Typst?

Several ways of creating notebooks for VEX exist, with most adopting visual editors such as google slides or hand writing their notebooks.
When deciding what we wanted to use for this season, we quickly ruled out hand writing the notebooks as mistakes could take valuable time to correct; neatness and clarity is often sacrificed; and the need for online collaboration is great. We previously used google slides with good results, however the formatting (e.g. colour coding, table of contents etc.) takes a significant ammount of time to maintain and can be very difficult to keep consistent when we all share equal role in notebook creation (as opposed to 1 person overseeing all notebook formatting).
We then landed on the possibility of using a markup language; and with the lack of flexibility from LaTex, Typst seemed like the best option. We had also noted a few teams success with using Typst, especially when using Notebookinator alongside it - for example team 53E (also the creators of Notebookinator) had a great Over Under notebook[1] using Typst.

Features

- Uniform formatting
- Notebookinator template
  - Easy cohesion with engineering design process
  - Built in components i.e. pros/cons tables
- Code blocks
- Built in table of contents
- Fully Digital
  - Neatness
  - Modern tooling
  - Easy submission
  - Cohesion with version control

[1] *Link to notebook*

The Snowflakes' Enginnering Ethos

Our Engineering Ethos

At its core, VEX Robotics is nothing but an engineering problem. It provides a goal, and the materials to get there. We believe that the key to success in Robotics intrinsicly lies in how you approach each problem; with open-mindness and the willingness to learn but most importantly the deterination to find the best solution possible.

> Engineering: The art of organizing and directing men, and of controlling the forces and materials of nature for the benefit of the human race.
>
> — Henry Gordon Stott

Our Engineering Design Process

For every new problem, we try to stick to an engineering method (similar to a scientific method) where different phases are used to maintain organisation. The process applies to all forms of design, including programming and sometimes even tactics.

Phases of Design

For each of the phases in our EDP, a corresponding icon is provided, these are used throughout the notebook to label a phase.

Identify Problem[1]
Each solution starts with a problem, this ranges drastically – for example, from 'We need a drivetrain' to 'this mechanism causes instability'. Problems can be raised by all members and, regardless of severity, it is something that must at least be addressed.

Brainstorm Solutions
Once the problem has been properly analysed, with root causes found, the team can move on to brainstorming solutions. Here, every team member can put forward possible solutions or fixes to the problem at hand, this is often accompanied with rough concept drawings. Additionally, finding use cases where each possible solution has been used effectively can be key to display a solution's viability.

Decide Solution
Once all possible solution have been brought to the table, one possible solution is picked to move to the next phase; to ensure that the decision is definitely the best one available, additional proceses can be used to decide the best (i.e. decision matrices). Ideally, all members offer their thoughts on the solutions.

Implement Solution
With one solution in mind, the solution can be expanded, now taking into account the smaller details while creating a plan of action for the designing and implementation of the solution. The solution is then designed and built – either physically or as a program.
Note that both the build and program icons are used during this phase.

Test Solution
Once the solution has been implemented onto the robot, we can begin testing the solution to find out how effective it is. This is a key phase as it shows us how the solution up to different scenarios.
Depending on its effectiveness, the results of a test may prompt us to move back into the implement phase, as changes sometimes have to be made. This creates a feedback loop that iteratively improves the solution until it meets our desired standards.

For our robot, we decompose the larger problem into a set of smaller, approachable problems. From there, each and every problem is approached using this EDP; this allows us to stay organised and avoid decision paralysis.

[1]Sometimes, if the problem is obvious (i.e. the need for a drivetrain), this phase is skipped due to mutual understanding.

Innovate Submission

DATE: 27.09.24

Novel Aspect Description:

We believe that our in depth and descriptive consideration of the possible uses of different sensors is particularly innovative. The thought and engineering research that went into it is rounded and well developed, with almost every combination described – with pros and cons and special considerations.

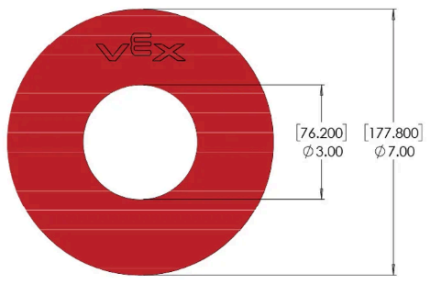
Pages:

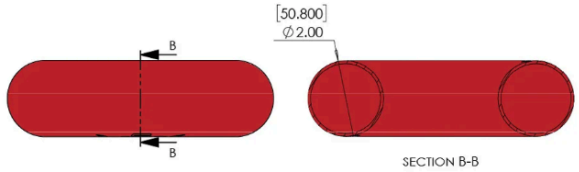
From page 46 to 58 (see table of contents).

The Game

(All images found from the manual [1])

Scoring & Game Objectives



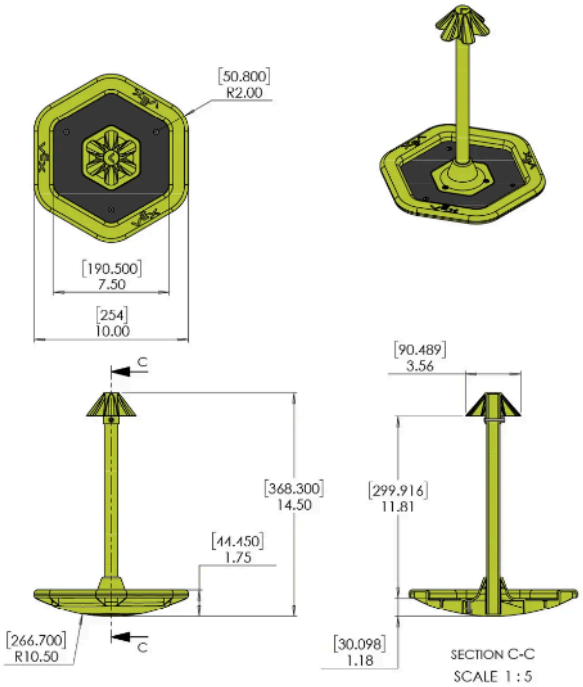


Game Elements: Rings
The majority of the scoring is done via these coloured rings.

- Outer diameter 7”
- Inner diameter 3”
- Height 2”

Potential Challenges

- Rings cannot roll on the floor, so manipulation is strictly contact based
- Large surface area in contact with floor so potential difficulty in manipulation



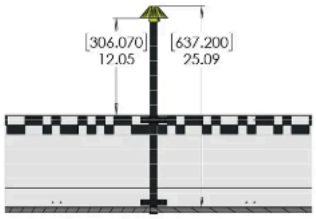
Scoring Object: Mobile Stakes/Goals There are 5 mobile goals (‘mogos’) on the field, and they can be freely manipulated by teams.

- 10” diameter Hexagonal bird’s eye view profile
- 14.5” height
- Flexible rubber cap

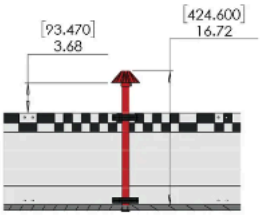
Potential Challenges

- Weighted bottom could make manipulation difficult
- Scoring would require an elevated mechanism.
- Rubber caps means that force is required to score/descore

NEUTRAL STATIONARY GOALS:



ALLIANCE SPECIFIC STATIONARY GOALS:



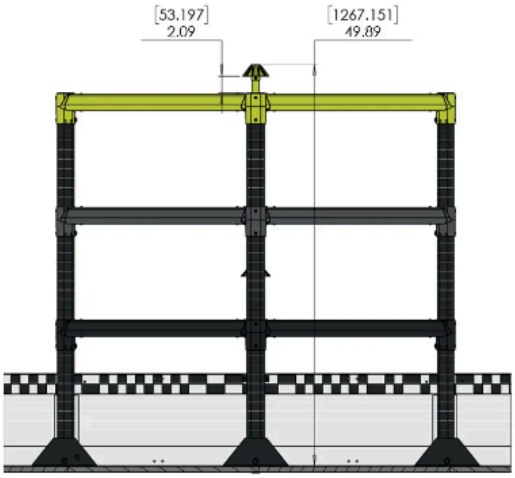
Elevated Stakes: Neutral and Alliance

There are 2 neutral stakes and 2 alliance stakes allowing for further scoring.

- Neutral stake 25.09" tall
- Alliance stake 16.72" tall
- Rubber caps (identical to mogo)
- Alliance stakes can only be scored by the corresponding alliance

Potential Challenges

- Stakes differ in height from each other (also from the mogo) meaning different or morphing mechanism to score on all.
- Placement (field perimeter) risks throwing rings out of the field (risking S1 infringement)
- Rubber caps mean force must be required to score/descore



High Stake and Ladder

In the center of the field, there is a 4' ladder that teams can climb in the endgame to gain extra points. It also has a stake that can fit 1 ring at the very top.

- 49.89" (4.165') tall
- 3 tiers/rungs
- 4 sides

Potential Challenges

- Climbing structures requires lots of power and/or time
- High Stake would require extreme precision

Scoring Takeaways

- All scoring requires vertical capability
- Employing multiple methods of scoring (mogo, neutral/alliance stake) would require multiple systems or 1 complex system such as a lift
- Emphasis must be put on **precision** and **reliability** as there is little room for error

Designed by: Daniel Dew
Witnessed by: Jonah Fitchew

Rules Analysis

Scoring Table

Ultimately, scoring can be summarized into one table, called a scoring table. While it will not allow us to gain insight on the specific intricacies of each rule and the challenge it presents – it can allow us to quickly see what aspects of the game score more points, therefore allowing to adjust our game strategy.

Rule name	Points scored	Associated Rule
Autonomous Bonus	6 Points	*<SC2>*
Ring Scored on Stake	1 Points	*<SC3>*
Each *Top* Ring on Stake	3 Points	*<SC4>*
Ring Scored on High Stake	- (Ring is considered as the 2 above, and may apply bonus as seen in <SC9>)	*<SC9>*
Climb Level 1/2/3	3/6/12 Points	*<SC7>*

Format

To avoid simply regurgitating the rules (to people who already understand them), we are going to simply list some rules with a paraphrased description; then how it affects us; then potential solutions – if a rule presents no problem, we will not cover it.

e.g.

<RULE NUMBER>

- *Paraphrased rule description*

Problems

- This rule affects us like this
- It also affects us like this

Potential Solution

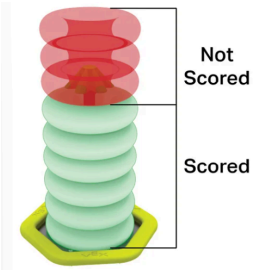
- This is one way we can mitigate the risk of infringement…
- This is another…

> **Note**
> Inspection, safety and general rules will not be covered, due to their relative simplicity.

Scoring rules

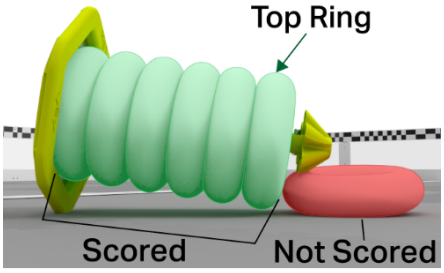
<SC3>

- To be considered scored on a stake, the ring must meet certain criteria:
  1. Ring must not be contacting robot *of same alliance*
  2. Ring is not contacting foam tile
  3. Ring is encircling the stake[1]
  4. Total ring count must not exceed max ring count of the stake (mobile & neutral: 6, alliance: 2, high: 1)



Problems

- Mogos with our rings on can be tipped to effectively descore some rings



Potential Solutions

- Driver may have to take care when scoring on neutral/alliance stake
- Driver may have to guard or defend filled mogos

<SC5>

- A mobile goal is considered placed in a corner when it meets the following criteria:
  1. Mogo is contacting floor/foam tile
  2. Mogo is upright
  3. Contact with robot is irrelevant

> **Note**
> Only 1 mogo can be considered placed in each corner, even if 2 meet the requirements.

Problems

- Mogos can be knocked over to mitigate effect of corner

Potential Solutions

- Driver can guard/defend the corner, especially as robot contact is irrelevant.

<SC6>

- A mobile goal that has been placed in a corner will result in the following modifiers being applied to its scored rings:
  - Placed in **positive** corner:

[1]Long description omitted, see *https://www.vexrobotics.com/high-stakes-manual#sc3*

1. Values of all scored rings will be doubled

- Placed in a **negative** corner:
  1. Values of all scored rings will be set to 0
  2. For each ring, an equivalent amount of points will be effectively removed from that alliance's score
  3. Points scored from auton bonuses and climbing cannot be removed

Examples included *here* .

Problems

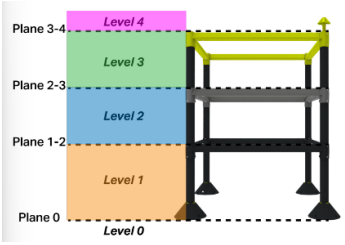
- Opposing rings scored in a positive corner can drastically change outcome of game due to 2x multiplier
- Ring scoring can be easily countered by placing them in negative corner

Potential Solutions

- Once again, large emphasis must be placed on defending scored rings and preventing them from being placed in a negative corner
- Putting emphasis on scoring on the elevated stakes could mitigate dependance on mogo scoring and corner defence/offence

<SC7>

- A robot has climbed to a level when the following criteria is met:
  1. Robot is contacting the ladder
  2. Robot is not contacting any other field elements
  3. Robot is not contacting any mobile goals
  4. The robot's lowest point is above that level's minimum height



Problems

- Climbing must be completely independant, it cannot rely on lower rungs or the floor

Potential Solutions

- When considering climbing, large power consumption – due to independant climbing – must be considered, possibly with use of a winch and/or a PTO[2]

<SC8>

- **Autonomous Win point** is awarded to *any* alliance that have completed the following tasks (as long as they have not broken any rules):
  1. At least 3 scored rings of that alliance's colour
  2. A minimum of two 2 stakes on the alliance's side of the autonomous line with at least 1 ring of the alliance's color scored
  3. Neither robot contacting or breaking plane of alliance's starting line
  4. At least 1 robot contacting ladder

Problems

[2]PTO: Power Take-Off

- Even if we can complete as many tasks as possible, AWP is still alliance teammate dependent because of criteria no. 3

Potential Solution

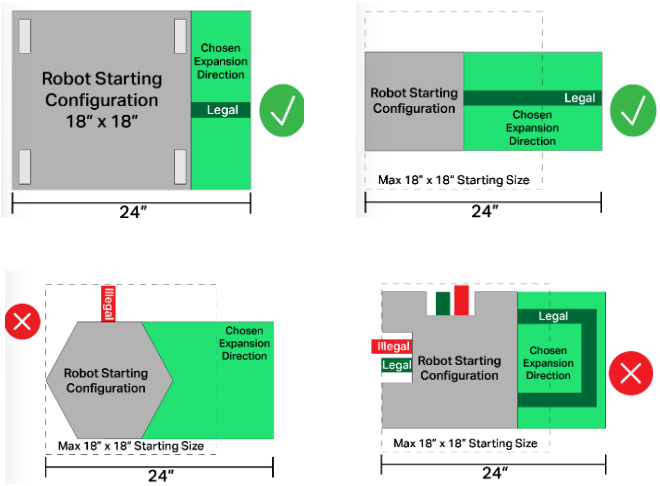
- Ensure prior coordination with teammate to ensure that they move off the line at the start[3]

Specific Game Rules

<SG2>

- Horizontal expansion is limited to an <u>additional</u> 6" on **one** side.

> **Note**
>
> 6" expansion is based on an 18" x 18" starting size, therefore robot can expand to the limit in **1** direction, then 6" in the same direction.



Problems

- Mechanisms that rely on expansion must be contained within the footprint of the robot, or not expand over 6" on one side only

Potential Solutions

- Design all expanding mechanisms to expand on one side only
- Use as little space of the 18" x 18" to maximise expansion capability

<SG3>

- **Vertical expansion is limited**; vertical expansion cannot break **2 or more** levels of the ladder

[3]If the team does not have a (working) autonomous, advice/technical help can be given to simply move off the line, ensuring AWP

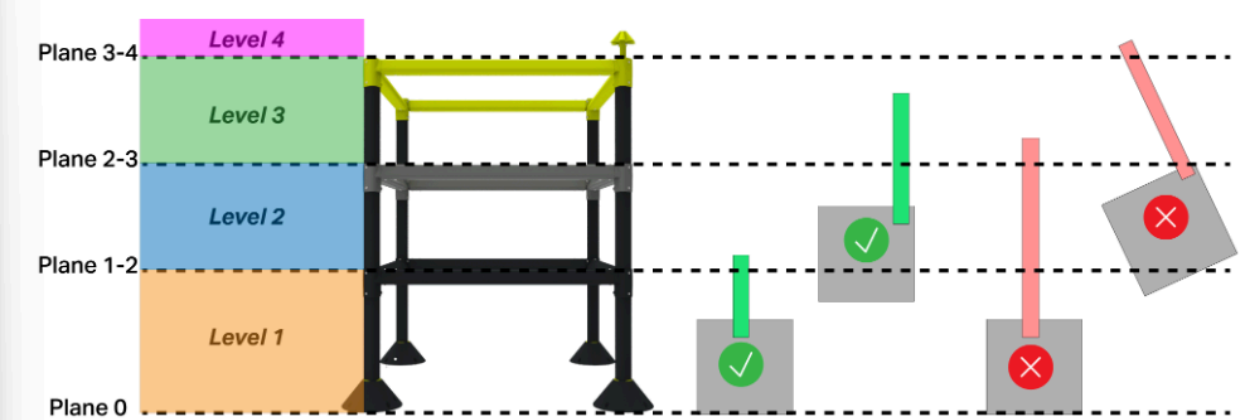


Figure SG3-1: Examples of legal and illegal vertical expansion.

Problems

- This rule makes climbing to the top with 1 movement impossible – unlike in Over Under – instead teams have to climb the ladder like a ladder, using each rung and not skipping levels

Potential Solutions

- When designing climbing mechanisms, multi-stage movements must be incorporated; making sure that the robot does not break 2 or more planes no matter the rotation[4]

<SG4>

- Keep scoring elements **in the field**, rings that exit the field will be given to the corresponding alliances to reintroduce into the game.

Problems

- Accidentally removing rings from when, for example, scoring on wall stakes results in a minor violation

Potential Solutions

- Driver can take extreme care when attempting to score on wall stakes
- Line-up guides can be designed to aid the driver
- Lots of time on tuning the mechanisms to ensure they are not too powerful

<SG6>

- Possesion is limited to 2 rings and/or 1 mobile goal
  - Where rings scored on a stake do not count towards possesion count
  - Plowing multiple mobile goals is legal only when no mobile goals are possessed

Problems

- When attempting to rapidly score rings, this rule may be broken due to more than 2 rings being possessed
- Accidentally plowing a mogo while possessing one will result in a violation

Potential Solutions

[4]This is because the planes are measured from the perspective of the field (see long explanation *here*)

- For both problems, driver care can be applied to avoid SG6 infringement
- A distance/colour sensor could be used in conjunction with an algorithm to stop manipulating rings once at the possesion limit
  - Using a colour sensor could allow for a colour sorting algorithm to only intake alliance's rings

<SG7>

- Dont cross the autonomous line during autonomous
  - Robots must not contact or break the plane of the autonomous line[5] during autonomous

Problems

- Accidentally crossing line due to lack of tuning or planning of the autonomous movements would result in the loss of ABP and AWP

Potential Solutions

- Extreme care and consideration must be used when planning out the autonomous movements

Primary Takeaways

Certain solutions appear more than once, meaning we can **prioritise them** to mitigate more risks at lower time/complexity costs.

Driver Skill

We have concluded that driving is a factor in nearly all the rules especially those targeted for defence and offence. High Stakes is a skillful game that requires lots of practice from the driver. Putting emphasis on training our driver, using drills, friendly matches etc. must be a priority.

Control and Precision

We have also concluded that **precision is key** to avoiding rule infringement and also to maintain effectiveness. All mechanisms must be designed with extreme precision with lots of time allocated for fine-tuning in order to:

- maximise effectiveness of mechanism
- avoid breaking rules such as SG4 and SG7.

The Plan

This game and rule analysis has allowed us to form a plan on how we will approach the coming weeks as we organise ourselves to tackle the season.

Timeline Considerations

- We will place emphasis on having time for plenty of driver practice

[5] basically the halfway line

- We plan to spend lots of time tuning the autonomous, in order to get it **consistent**, which means we will have to make ample room for autonomous testing in the timeline

Careful Design

- We will also be making sure that all our designs are designed with **strength, precision and effectiveness** in mind during all stages of the design process – this is especially prominent during the CAD phase. Keeping these three factors in mind will ensure that the systems utilised in our robot are as **competitive** as possible.

Teamwork Wins Championships

To ensure that we achieve our maximum, we must organise ourselves into a team that works hard to solve the problems presented to us. We can do this by sorting ourselves by roles and skillsets.

> **Quote**
>
> Talent wins games. Teamwork wins championships.
>
> — Michael Jordan

Goals & Objectives

Before we begin to design and build a robot for High Stakes, we must first define clear goals of what we **as a team** want to achieve. This will allow all members to work towards common objectives.

Our Goals For High Stakes

- A Force To Be Reckoned With
  - We want to become a team that is competitive and 'formidable on the field'
  - We can quantify this goal mostly by how high we place in qualifiers – we want to aim for top 5 seed in most competitions
- Nationals
  - We also would like to qualify for the UK National Championship
  - We achieved this goal last season so would love to get this oppurtunity again.
- Worlds/Internationals
  - This is a very difficult goal to achieve, with only ~5 spaces from the UK
  - We were also good enough to get this oppurtunity last season, with us going to the 2024 World Championships in Dallas

Organising Ourselves

Roles

We can first organise ourselves by our skillsets; we can do this by asigning roles[1]:

- Head Tactics: Thomas Robb
- Programming Support & Notebook: Daniel da Silva
- Head Programmer & Driver: Daniel Dew,
- Co-Head Builder: Aubert Seysses
- Co-Head Builder: Jonah Fitchew

[1]Roles can also be seen in the *Team Introduction* pages

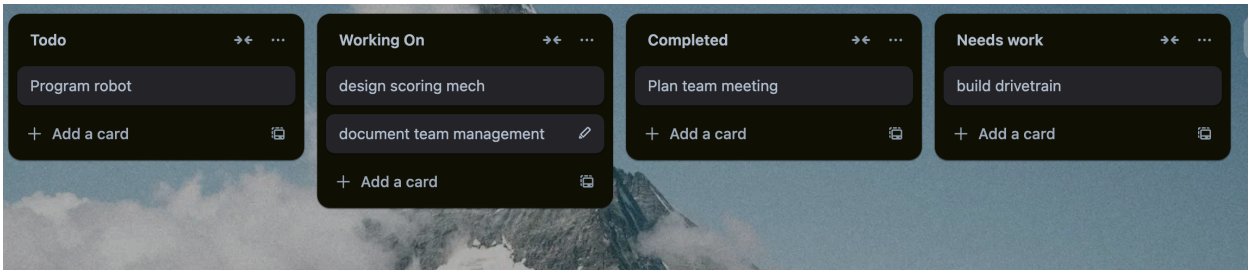
Organising ourselves like this ensures that when a problem arises, we all know how to approach the challenge with our unique skillsets; it can also be helpful when introducing the team, as it helps judges and peers understand what we do within the team.

Project Assignments & To-Dos

Throughout the season, there will be lots to do over various projects – from designing a new robot (or subsystem), to writing a couple lines of code. To manage all these tasks, we will generally asign a project or to-do to a few members – using the roles as guidelines (e.g. building tasks asigned to builders, programming to programmers etc.); progress on said tasks will be discussed in in-person meatings[2], or using our channels of online communication.

Trello

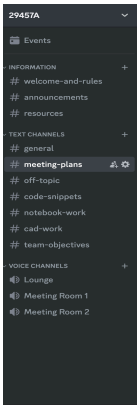
Trello is a project management software that allows teams to create dynamic to-do lists. 'Cards' can be dragged into different columns to display their state (e.g. 'todo', 'Working on' or 'completed'), we can asign members to certain tasks, give them deadlines, or add a checklist for each. We found it to be exceptionally useful last season, and we plan on using it for this season.



An example of a trello todo list

Discord & Online Communication

As a less formal method of communication, we use both Discord and Whatsapp to discuss team objectives and progress. Discord is especially helpful as it helps us to organise our chats into channels (see right). Another useful application to discord is sharing files and resources amongst ourselves – for example we can share small code changes on the discord for testing.

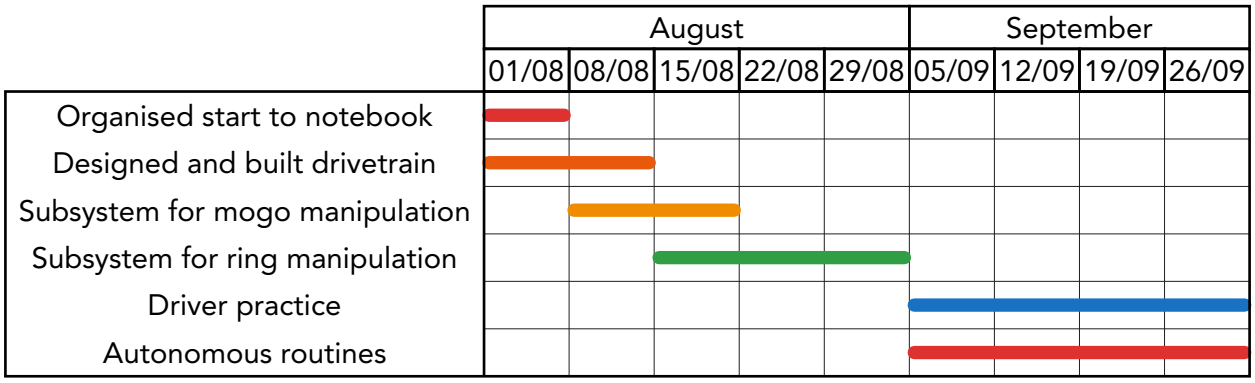


[2]Due to holdays, no in-person meetings have happened yet – our first full in-person meeting is scheduled for 03/09/24

Early Season Projects

We have a list of goals we want to acomplish for early season:

- Organised notebook
- Designed and built drivetrain
- Designed and built subsystem for mogo manipulation
- Designed and built subsystem for ring manipulation
- Substantial amount of driver practice
- Reliable and semi-complex autonomous routines



Ensuring quality

One of the most important things to consider when building a robot is its Build Quality. This is really important to us as the better our build quality; the fewer repairs which we will need to do mid competition. In addition, better build quality has a positive correlation with competitive competition performance. As a team we have decided that this year ensuring good build quality is paramount in our considerations when constructing each robot this season. This essentially comes with a set of principles for how we should build a robot that we should adhere closely to in order to construct a robot that can be brought to multiple competitions, as the fewer rebuilds we do the more time that we'll have to tune and practice.

Our building principles

As previously mentioned we are abiding by a set of rules, which have the intention of improving build quality.

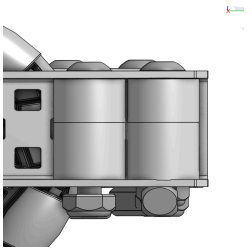
Boxing

The first of these practices is **boxing** parts of the robot, which is used to provide structural integrity. There are two main types of boxing:

- Spacer Boxing
- Keps Boxing

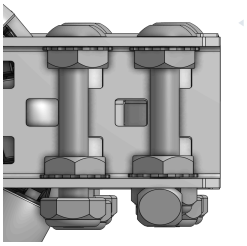
Spacer Boxing

This image taken from the CAD of a previous drivetrain is a good example of boxing. Spacers are placed in the gap in the C-Channel to prevent deformation. As when tightening screws its quite easy to bend the flimsy aluminium flanges, which can cause friction and many other complications should it happen throughout the robot build. By doing this to our joints we increase rigidity and structure to joints and so minimise bending.

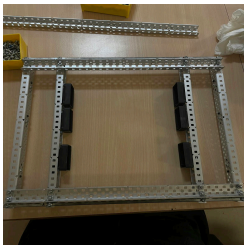


Keps Nut Boxing

Spacer Boxing isn't always possible because elements of the robots design may get in the way - such as bearings - so in the event of this we can use Keps Boxing instead. This provides some of the same functionality that boxing does with the addition of keeping space parts such as bearings. It utilises a screw with keps nuts fastened to the top and bottom of the C-Channel with a nylock on the bottom that holds it all together. This can be used to make the structure more rigid, which is important for things such as braces.



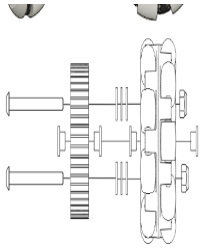
Squaring



Squaring a drivetrain involves temporarily bracing the main C-Channels to hold them in place. This allows for precise measurements to verify that all angles within the chassis are 90 degrees. Subsequently, permanent bracing is added, ensuring that all C-Channel are held so that they are perpendicular or parallel to each other. This meticulous process guarantees proper bracing alignment, which helps minimise friction within the drivetrain.

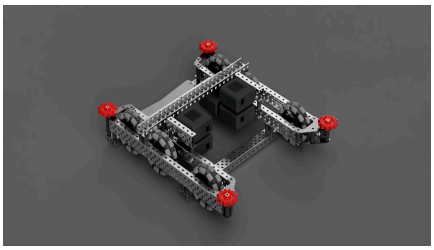
Screwjoints

To minimise friction and improve rigidity in rotating joints (like floating intakes or drivetrain wheels) that don't require motor power, we will prioritize the use of VEX screws over axles. VEX axles are prone to bending, causing increased friction. VEX screws offer superior rigidity and lower friction in these applications. Additionally, they provide a convenient point for indirect bracing, further enhancing the structural integrity of the robot. Below is an example of screw joints being used in a "wheel pod".



The CAD Process

Another thing we would like to improve on this season is the use of CAD when designing our robots. Last year we did a lot of "freebuilding", which is where we go in with an idea and just start building. Instead this season we will use CAD (Computer Aided Design) to do the majority of our design and prototyping before reaching the physical build stage. This minimises janky solutions that employ the "if it works don't fix it" ethos and allows for a more economical and efficient use of parts and the construction of an overall cleaner robot. After CADing, we will prototype each mechanism in order to iron out any kinks before implementing them into them into our final design as it will be more difficult to make adjustments to mechanisms when fully integrated into the robot. Finally, we will build the finished CAD as a robot, which will be ready for testing beyond .



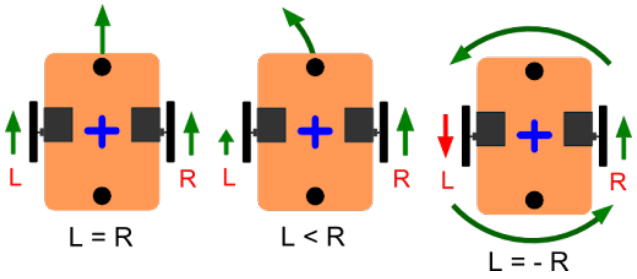
Testing our Robots

Testing is a crucial part of the process in creating a working robot because up until you get to metal you have almost zero idea how the robot will interact in a real world environment, so during our building we will employ testing procedures and make necessary adjustments to ensure that the final product truly is the best it can be and there is no situation where it doesn't do what we planned. This process is called tuning. the testing practices that we will be using include pneumatics testing which will ensure through the constant usage of the pneumatic mechanism that it uses little enough air to last through a match of normal usage. motor testing is also paramount for match longevity, we will all the robotsd motors until an overheat message is observed on the brain to ensure that the motors work at peak performance for "the greatest 2 minutes of robotics" that we need them to. finally, by testing our robot against our sister teams, 29457B and 29457X we can ensure that it is rigid enough to withstand the vicious nature of a high stakes match.

Designed by: Snowflakes
Witnessed by: Daniel da Silva

The main types of Drivetrains:

In VEX there are two main drivetrain archetypes, which we can we use to solve the fairly obvious problem, that we need to be able to move[1].
The first type of drivetrain is the Differential Drive, which features mirrored sides in a typical "tank" formation and allows for independent manipulation of each side, to allow for forwards and backwards motion, turn on the spot or turn left and right whilst moving forwards and backwards. Below is an example of the configuration of a Differential Drive.



From this website

The second main type of drivetrain are the Holonomic Drives. In this category, there is the Omni Drive, X Drive and the Mecanum Drive. The Omni Drive features one omni wheel in the centre of the drivetrain and allows for strafing movement as well as the conventional movement allowed for by a differential drive. Third is the Mecanum Drive, which takes advantage of the VEX mecanum wheels, which allow for strafing and a limited form of horizontal movement.

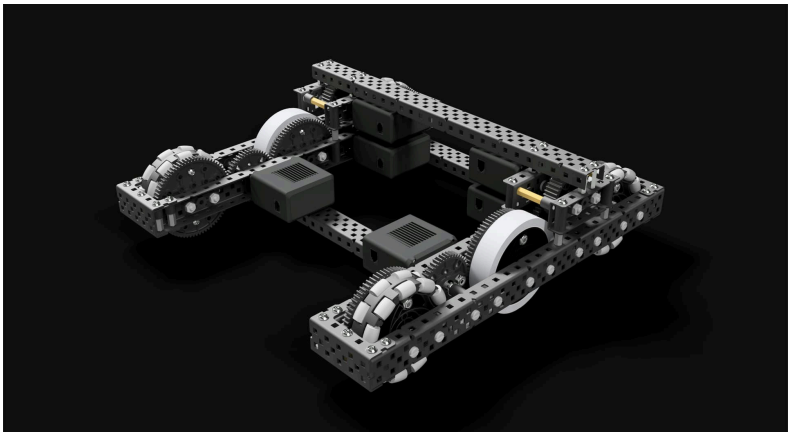
Differential Drives vs Holonomic Drives

We shall compare the drivetrain archetypes first since the different Holonomic Drives share a lot of similarities and in practise there is very little difference in terms of layout between differential drives.

Differential Drive

Pros	Cons
• Simplicity - Easy to build, program, and maintain • Powerful - Excellent for pushing and traction-heavy tasks • Stability - Will be balanced and support all sorts of mechanisms	• Limited manoeuvrability - Only supports forward, backward, and turning; no strafing motion • Not agile - Slow to make fine, precise movements or quick direction changes

[1]The typical 'identify problem' page is omitted since the ability to move efficiently is non-disputable.



Differential drive example from this forum thread [2]

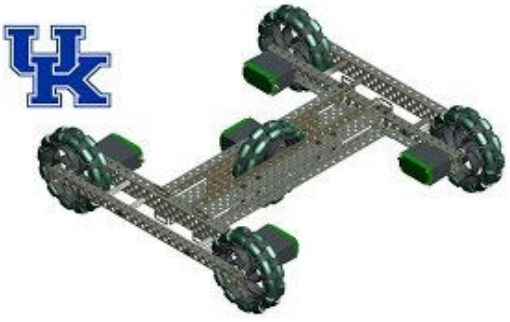
Holonomic Drives

Pros	Cons
• Holonomic Movement - Can move in any direction (forward, backward, sideways, diagonal) with ease. • Manoeuvrability - good for accurate and small adjustments • Agility - Quick, fluid directional changes without rotating the robot.	• Lower traction - Limited pushing power due to because they have more complicated wheel layouts • Complex programming - Requires more advanced coding for full control.

Holonomic Drivetrains

Omni/H Drive:

Omni/H Drive is probably the most simple to build of all holonomic drivetrains with a single strafing wheel in the middle relying on the omni wheels rollers to provide lateral movement. The gear ratios used are the same as in a differential drive but one of the losses coming from the middle wheel is that spot is typically used to have an odometry pod to provide a higher level of accuracy during autonomous. It also means that one has to place the pneumatic tank higher up than ideal due to the need to brace and support the middle wheel using additional pieces of C-Channel.



Brainstorming Drivetrain 2024/08/04

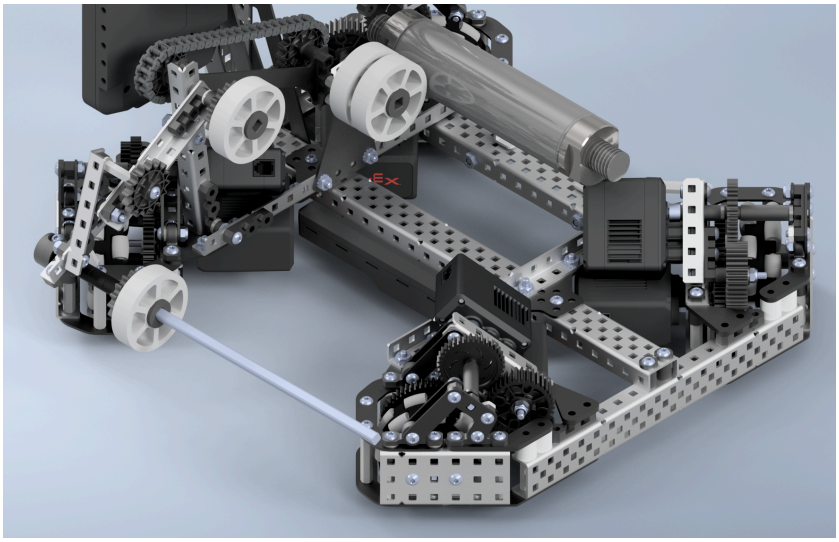
CAD model of a H Drive from Purdue Sigbots made by Kentucky University [3]

Pros	Cons
• Simplicity: The H Drive is easier to build and program compared to more complex holonomic drives like Mecanum or X Drive • Strafing ability: The central wheel allows for strafing movement • Efficient space usage: The parallel wheel layout in H Drive leaves more space for other components on the robot compared to an X Drive	• Limited lateral power: central strafing wheel in H Drive has less traction and pushing power compared to the Mecanum or X Drive, leading to weaker side-to-side movement • Less agile: H Drive doesn't handle diagonal movement as smoothly as X Drive or Mecanum, since its motion is less accurate and precise • Dependence on central wheel: If the central wheel fails in any way we lose the ability to strafe, which is the principal advantage of H Drive • Programming complexity: is high but less so than X Drive • Space: central wheel means less space for other subsystems such as odometry pods

X Drive:

X Drive has traditionally been used by teams for extremely precise programming as it can provide an edge over more traditional drivetrains in terms of accurate movements in things like autonomous skills. However in competition settings, X Drive isn't very popular as its more difficult to control. X Drives use a compound gear ratio to get around the difficulties in making the wheels diagonal, this can mean that it becomes quite difficult to build and maintain in comparison with more traditional drivetrains. Trying to fit the brain, pneumatic tanks and other subsystems around this drivetrain can be difficult due to the odd motor placements.

Pros	Cons
• Faster: than both Mecanum and H-Drive, as it naturally provides more efficient power transfer for quick movements due to the 45-degree wheel orientation • Simpler: (compared to Mecanum Drive) to build and program, since it doesn't require complex motor tuning or algorithms for strafing • Better diagonal movement: compared to other drivetrains	• Lower torque: X Drive generally lacks traction and pushing force compared to Mecanum, which can cause it to struggle • Inefficient layout: the diagonal wheels takes up a lot of space, which limits available area for other mechanisms compared to more compact configurations like H-Drive • Complex programming: but less so compared to Mecanum • Complex design: difficult to CAD

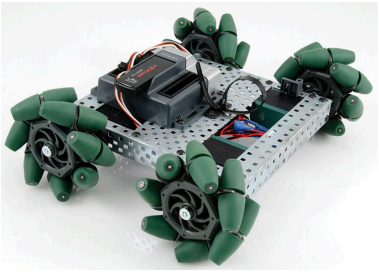


Here we see a render of an XDrive done by in_ithica | 3818 on the VEX CAD discord server [4]

Mecanum Drive:

The mecanum drive is one of the more challenging options for a team to invest in. However, mecanum drives have excellent manoeuverability, which can be incredibly useful in both matches and in skills challenges. One of the main building challenges this drivetrain offers is that it is much harder to gear since the size of the mechanum wheels means that it is impossible to maintain a 3 or 4 wide hole between the two drive C-Channels since mecanum wheels take up a lot more space. This means that you have to use direct drive gearing if you go for a mecanum drive.

Pros	Cons
• Holonomic movement: Mecanum wheels allow for full omnidirectional movement, allowing for orthagonal and diagonal movement and also strafing. • High traction: Compared to other holonomic drives, like X Drive or H Drive, Mecanum has relatively good traction • Versatility: Is useful in both skills and in matches conditions, due to its many movement options	• Complexity: much more precise motor control and alignment is required for effective movement • Power loss: angled rollers on mecanum wheels means some power is lost during lateral movement, making it less efficient compared to a tank or X Drive • Programming complexity: significantly harder to program



Model of a Mecanum Drive from the Servo magazine website about holonomic locomotion [5]

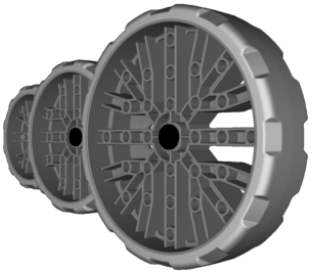
Wheel type and configurations:

Another consideration in our drive is the amount and type of wheels used. We can either have 4, 6 or 8 wheels on our drive and we also have a mix of omni and traction wheels or have a drive that's just traction or omni wheels.

Use of Traction Wheels:

For our drive we must decide whether we want to include traction wheels and the number and positioning, which we want to use. Having more traction wheels increases our grip but also reduces our ability to skid, which can mean that our bot will struggle to do tight turns quickly. However, not having enough traction wheels means we may be easier to push around and struggle to contest corners.
To maximise grip whilst retaining agility, it is common to use traction wheels in the centre to reduce unwanted pushing forces, whilst combining them with other wheels such as omniwheels on the edges for greater turning ability.

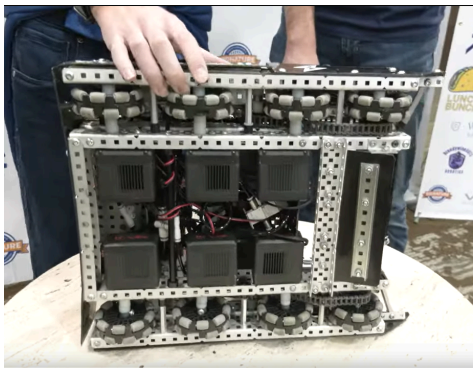
Pros	Cons
• Increased grip: Which means that you have more power. It's easier to push game elements or other bots around. • Resistance: Since they have a single degree of freedom, traction wheels make our robot more resistant against being pushed when attacking our defending	• Turning: a full traction drive will struggle a lot when turning, which isn't ideal



Traction wheels from BRLS Wiki [3]

Brainstorming Drivetrain 2024/08/04

All Omniwheel Drive:



Having our drive consist of only omniwheels has some advantages such as allowing for us to have “drift drive”, which allows for more control over the robot - if your driver is more skilled. However, a lack of traction wheels means our bot will be more susceptible to being pushed and stuggle in direct confrontations. This tradeoff can be compensated for with good driver skill in order to evasively outmanoeuver opponents. A good example of an all omniwheel drive was 9364H’s bot in OU[2] *Pits and Parts [6]*, who additionally engineered their bot for speed and agility to control the arena and punish double zoning in that game. See image on right for example (Image cropped from video cited above).

Number of Wheels:

Another important consideration is the amount of wheels, which we want on our drive. Similar to traction wheel ratio, having more wheel affects our torque-agility ratio. Having more wheels increases surface area in contact with the ground, which correlates with more traction. An increase in traction means that we will have less skidding, which can be a detriment since skidding can be useful for turning and allows for “drift drives”, but it will also mean that the power is transferred more efficiently, which allows us to have more rpm or torque. See explanation for this on the right.

Energy = Power * Time
E=Pt
Energy = Force * Distance
E=Fs
Therefore
Fs=Pt
P=Fs/t
P=Fv
Power = Torque * RPM

4 Wheels

Having 4 wheels is the minimum number of wheels a drivetrain can have. It sacrifices power for manoeuverability and is easier to implement since it doesn’t require motor stacking.

Pros

- Agility: Slides more so it has greater theoretical turning ability
- Easy to implement: Since there are fewer wheels there’s fewer gears to deal with allowing for more space for game specific components such as wings, intakes or lifts
- Lighter: Less components so robot will weigh less meaning motors have less load to move

Cons

- Less power: Fewer wheels so there’s less traction, which means less pushing power
- Lighter: Less weight so it’s easier to push around

[2]OU - Refers to Over Under 23-24

Designed by: Jonah Fitchew, Daniel da Silva
Witnessed by: Daniel Dew

6 Wheels

6 wheels is a balance between manoeuverability and power. However, it's trickier to implement since it requires motor stacking.

Pros	Cons
• Agility: Still slides but not as much as a 4 Wheel Drive • Power: Balanced power but still less traction than an 8 Wheel Drive	• More complex: Requires motor stacking so its design as a drivetrain is more complex, which can mean it's more difficult to implement game scoring subsystems.

8 Wheels

An 8 wheel drive provides the most traction of the three options, which gives it the most power but means it will slide less and so will struggle with turns more.

Pros	Cons
• Most Power: Has the most power because of the greater traction • Heavier: More wheels and more gearing weigh more so it's more resistant to pushing and will push with more force	• Less Agile: The higher traction stops the robot from sliding as much so it will have less turning ability. • Heavier: The greater weight will put more strain on motors, which leads to higher chance of motor burnout. However, this can be avoided by using hotswappable motors or cooling the motors with a fan.

Overall, it's important to consider the strengths and weaknesses of the wheels chosen and amount of wheels, since the power lost from using fewer wheels can be compensated by using some traction wheels to create overall high performance or similarly an 8 wheel drive can become more manoeuverable if the drive includes some omniwheels or is entirely made of them. Therefore by considering what the wheels and wheel amount can do for the drivetrain it is possible to create a drivetrain, that doesn't suffer from any major weakness within the scope of what our team is trying to achieve.

A Look at Gear Ratios

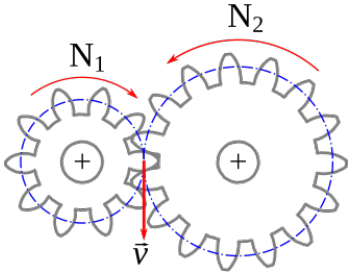


Image of a 2:3 gear coupling from wikipedia [7]

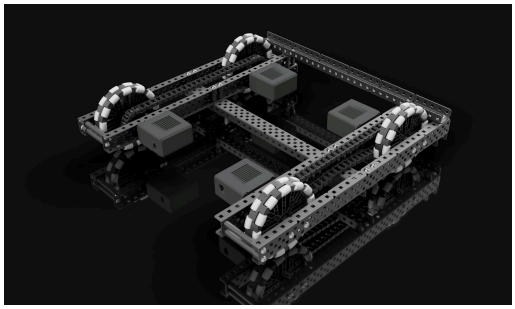
Another factor which contributes to the balance between torque and agility is the gear ratio chosen. If the force applied on the driven gear is further from its pivot compared to the distance from the driving gears pivot then the driven gear will turn with more force but it will turn slower compared to the driving gear. If this coupling is reversed then the driven gear will turn with less force but it will turn faster compared to the driving gear. The gear ratio is often

described as either $\frac{\text{driven gear } \Omega}{\text{driving gear } \Omega}$ where Ω is angular velocity or $\frac{\text{driven gear teeth}}{\text{driving gear teeth}}$. Like with wheel type and amount the gear ratio of the drive can be used to further optimise a drive to have greater speed or greater pushing power. It's also important to consider that wheel size also acts like a gear and so affects torque and speed.

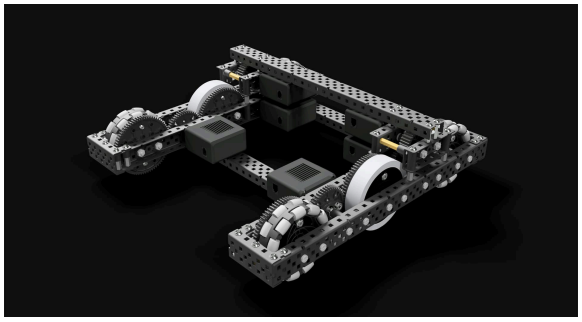
> **Note**
>
> Whilst it's not very commonly used in the context of V5, it's theoretically possible to have a gear transmission to switch between high torque and high speed, which would allow for dynamic gear ratio changing.

Direct Drives

It is also important to consider the viability of direct drives, where power is drawn directly from the motors. This can be useful as it has a lower build complexity allowing for more space and parts for other game scoring subsystems. They can be considered when you want to go with specific rpms such as 200 or 600, in order to have a unique torque-speed ratio compared to the rest of the competition. Below is a comparison between a direct drive and a 6 motor with some stacked motors.



Direct Drive [8]



Geared Drive [2]

Drivetrain Selection Process

The process of selecting a drivetrain is a teams first thoughts when considering the needs for a robot in any VEX season. The benefits and drawbacks of all types of drivetrain are considered at this point of the design cycle, in the context of the current game. Many different prototypes are considered since there a multitude of ways due to design a drivetrain because of the freedom provided by the rules. Naturally, not every solution is equal to others and so we must quantatively eliminate possibilities and decide on the best possible option.

Choosing drivetrain type

Holonomic drives and differential differ in terms of performance and we shall consider what would ultimately be best for us. The decision matrix shows our thought process.

	Manoeuvrak	Stability	Ease of build	Size of wheels	Possible gear ratios	Torque	Total
Differential	3	5	5	4	5	4	26
Holonomic	4	4	2	3	3	2	18

For our current purposes, a **differential drive** is the most optimal drive to consider using. This is because of their simplicity; which we feel is valuable this early in the season since most other drives are much more complex, which will ultimately make the designing of scoring systems more frustrating. In addition, differential drives provide good traction, which is helpful for pushing other bots, and their stability means we are less likely to tip over mid match, which is **catastrophic**. Differential have more gearing options than Holonomic drives. Furthermore, we have deemed that the downsides of a differential drive are either not substantial or can be compensated for by other parts of the drive. For example the lost manoeuverability can be made up for by incorporating omniwheels into our drivetrain.

Choosing Wheel Amount

Choosing the correct amount of wheels is an important consideration when designing the drivetrain. It is an extra way to achieve an optimal balance between torque and agility. The three main considerations for wheel ammount are:

- Traction
- Manoeuvrability

- Complexity

	Manoeuvrability	Traction	Complexity	Gear ratios	Total
4 Wheels	5	2	5	3	15
6 Wheels	4	3	4	3	14
8 Wheels	3	5	3	5	16

From our analysis we decided that an **8 wheel drive** is our best choice, since it synergises most strongly with a Differential Drive. Combining these two gives us great freedom for gearing, allowing greater control over torque and speed. Additionally, both elements will contribute to a higher power drivetrain. The simplicity of the Differential Drive compensates for the higher complexity of having 8 wheels. Unfortunately, neither of these design elements give us the greatest manoeuverability, which is the greatest problem with our design at this current stage. This can be addressed with the wheels we choose, since having 8 Wheels gives us the most flexibility over combinations of wheel types.

Wheel types

Going for an 8 wheel drive gives our bot great power, whilst also being relatively simple to build. The main consideration for the combination of wheels, which we choose is that they give our bot **greater manoeuverability** as that's the main problem with our current solution.

	Manoeuvrability	Traction	Total
All traction wheels	0	4	4

Mixture	3	3	6
All omniwheel	4	1	5

We have decided to go for a combination of 6 Omniwheels and 2 Traction Wheels, positioned third from the front. Having this wheel in this position means we still resist pushing from the sides but allows us to have greater manoeuverability thanks to the 6 omniwheels. Our experience from last season told us that placing the traction wheels there was best for turning and according to *purduesigbots* central placements helps reduce unwanted sideways movement. Another consideration was to place the wheels in a diagonal configuration, but experience told us that middle back was ultimately the superior choice.

Drive Gearing

The final consideration for our drive train will be the gearing we go for. This is the final way in which we can alter our drive train in order to get its torque and speed within desired parameters. We decided to use blue cartridge motors to give us 600 rpm, which we can gear down to 450rpm, with 3:4 gearing, to further increase torque but still have a good speed. We chose 69.85mm (2.75in) wheels to have greater power, so we don't compromise acceleration from the higher rpm. We didn't go for direct drive since we didn't want 200rpm or 600rpm.

Final Decision

For our Version 1 robot, we have decided to go for an 8 wheel differential drive using 6 omniwheels and 2 traction wheels. The traction wheels, will make up the third wheel pair from the front and we are going to run a 450rpm drive using blue cartridge motors. We believe that this combination will lead to a solution with ideal speed, acceleration and torque.

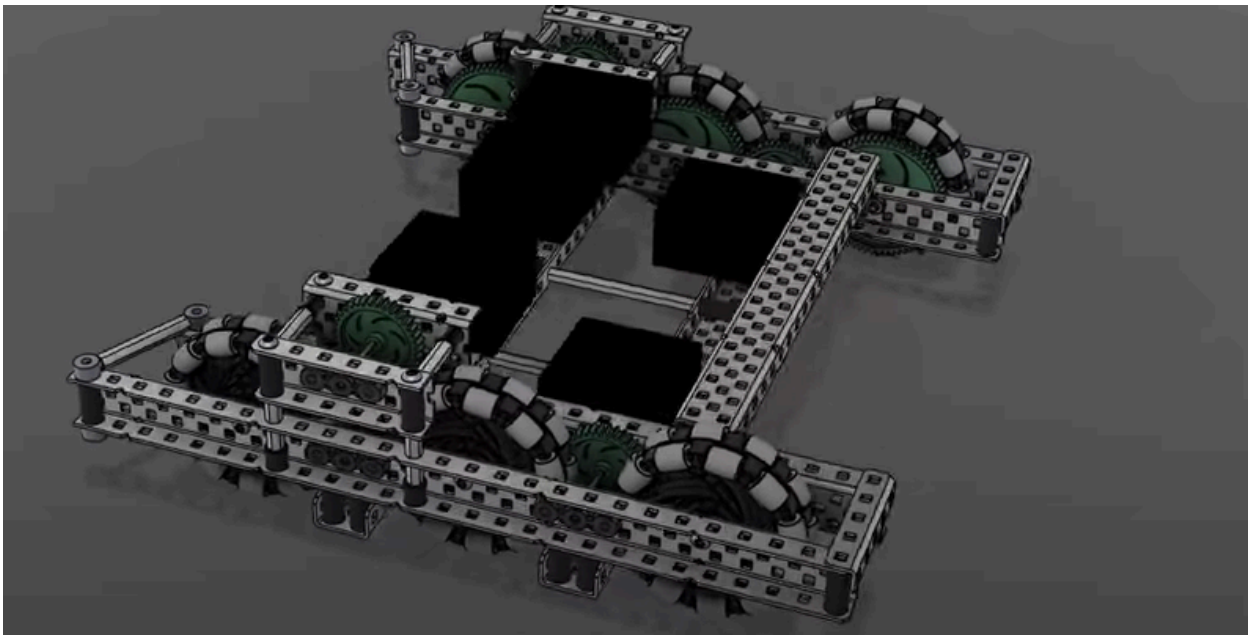
CADing the Drivetrain

Before building anything in VEX, we must first utilise Computer Aided Design software (CAD) to visualise what the robot is going to look like and whether or not the ideas we have will work. CAD allows us to use any and all VEX parts that can be bought, and it allows for outlandish ideas to be visualised on a screen before any time or money is wasted on something that will not be viable in competition.

Because of this wonderful resource, we always start any part of the robot that we aim to build in the virtual space, using *Autodesk Fusion 360* , which allows us to import the VEX parts library and take advantage of the range of features it offers.

Our Aim with CAD

Our aim is to have a fully virtual version of our drivetrain so that we can build it in real life with the closest accuracy possible and in the cleanest, most sustainable way. This should help us build an error-free robot and avoid having to rebuild.



An example of a finished CAD model of a drivetrain done by team 29295A [9].

Starting Off

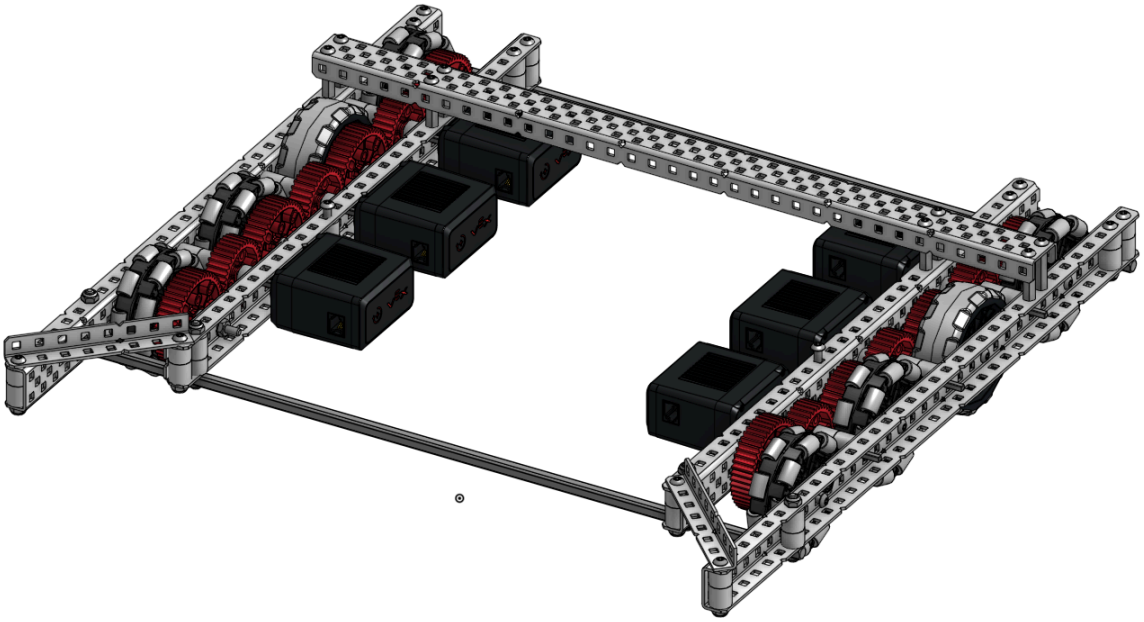
The simplest thing to CAD is probably a drivetrain, as it does not often differ much from season to season, being fairly evolved in its entirety. We have decided on the following specifications for our drivetrain:

- 450 RPM at a 3:4 gearing
- 8 wheels
- 6 motors in a differential layout

Therefore, it is fairly easy to create a general CAD model that could provide an idea of what to build with real metal. However, to create a complete replica of what is going to be built, one must spend a little more time on the intricacies of the CAD model, taking time to add each individual spacer and screw to ensure clarity when the time comes to build.

Our CAD Model

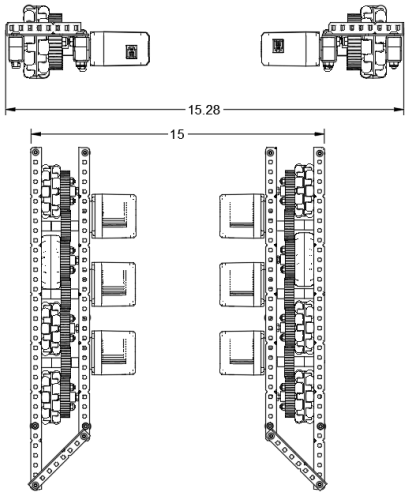
A few members of our team are fairly proficient in CAD, especially Fusion 360, due to needs outside of robotics. The process of creating a CAD model exactly to our wants and needs did not take very long.



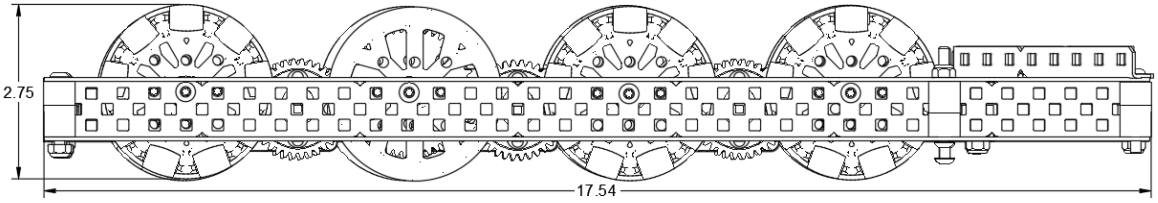
Here is the CAD model of our drivetrain, that we will use to build the drivetrain for our first iteration robot. This CAD model will help us with actual building and hopefully speed up the whole process.

Drivetrain CAD Drawings

Before starting construction on our drivetrain, we have created a set of engineering drawings, which show the measurements of the drivetrain in inches. This is helpful for design of parts that do not come directly from VEX, such as polycarbonate pieces, which we will use throughout the build process for custom parts serving purposes that VEX may not solve directly or inefficiently:



Here we see the top down view of the drivetrain (without bracing) showing the measurements in inches.



Here is the side on view showing the height of our drivetrain, which is fairly low to the ground, which will help us maintain a low centre of mass. Maintaining a low centre of mass is important because it means we are less likely to tip over and overall be more stable.

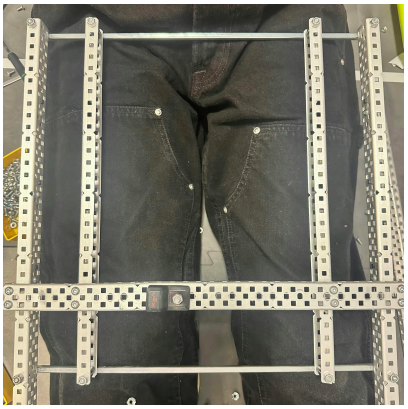
Building the Drivetrain

Step 1: Building the Chassis

The first step when building a drivetrain is to construct the chassis. This consists of the **4 C-Channels** that hold the wheels together, plus the bracing that keeps the C-Channel aligned in a perpendicular fashion, allowing for smooth driving with minimal friction.

Unfortunately, due to inconsistencies and tolerances within the screw to C-Channel contact, these 4 C-Channels are often misaligned, leading to friction and instability. To minimise these inconsistencies, we used a technique called **squaring** when building the chassis. This involves attaching the 4 main drive C-Channels to 2 other C-Channels to create a box, where all components are either perpendicular or parallel to each other.

We then used an engineer's square to ensure that each angle was **90 degrees**. Afterwards, we fitted the braces to ensure that our drivetrain remains straight since keeping these angles perpendicular will reduce friction as much as possible. After fitting these, we removed the temporary braces since they are no longer needed, leaving us with this:



Step 2: Screwjoints and Friction

Now that the drive is braced and sturdy, we can attach the wheels and test the friction - both for the wheels themselves and when they are attached to the gears in the motors. To do this, we used the function on the brain that allows us to see how much power each motor is pulling

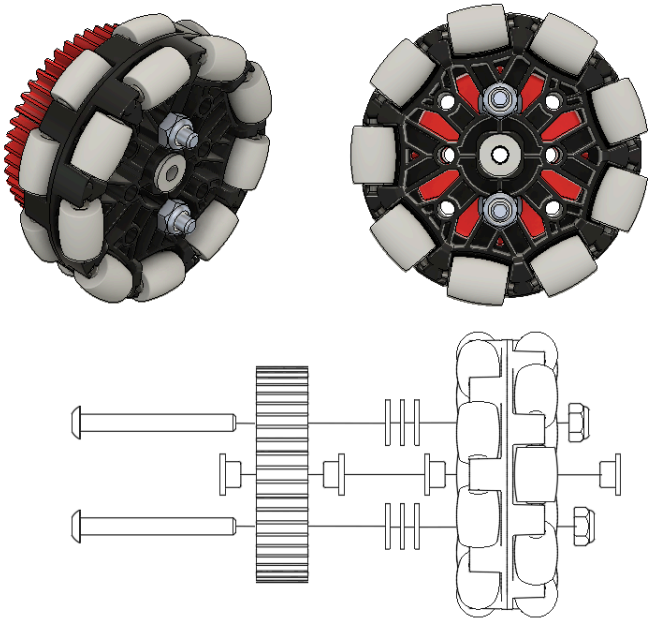
at any given time. Typically, a VEX motor outputs about **0.2W of friction** on its own, meaning that this is the target for friction for the entire drivetrain.

Reducing friction is important because high drive friction will limit the power output of our drive, which means that we effectively lose torque and speed, which is wasteful and ultimately undesirable. Therefore, having low friction is a high priority, which is a lesson we learned from last season as we struggled a lot with friction in last year's game.

To attach the wheels with minimum friction, we used something called a **screwjoint**, which utilises the fact that a VEX screw is much sturdier and less prone to bending than an axle. When working with a geared drivetrain, this is particularly helpful as it means that the motors do not need to power the wheels directly.

Step 3: Wheel Pods

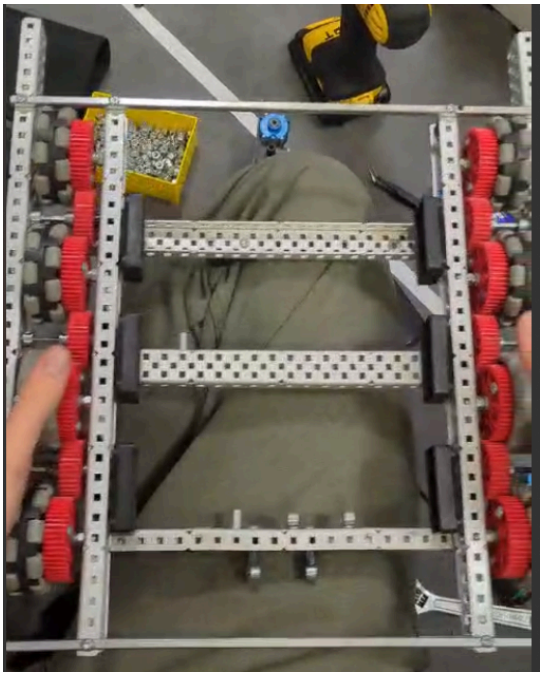
For this to work, we used the **VEX round inserts** in each of our wheel pods, which contain a 48t gear and wheel screwed together. On a screwjoint, they would otherwise spin independently.



As seen from this exploded view of the wheel pod, this is how our wheels come together with the gear to create a seamless geared drivetrain with minimal friction, utilising compact screw joints since there isn't enough room for a single bearing screwjoint.

Step 4: Attaching the Wheel Pods

The next step is to attach the wheel pods to the drivetrain so that it becomes driveable.



In this photo, we can see that the wheels are attached in conjunction with the **36t gears** that allow us to have **450 RPM** on our drivetrain. You can also see another technique we use when building, which is to have **hotswap motors**. This means they can be easily switched out in the event of a burnout during competition.

What this entails is that the motor itself, along with the gear cartridge, has been removed from the motor cap. The cap is fastened securely to the drive with screws, whereas the cartridges and motor are attached with a zip tie or rubber bands so they are secure during driver control.

Completing the Drive train

At this point, we had a drivetrain, which was fully braced with C-Channel and high strength shaft. Additionally, with all of the gearing on our drivetrain done, we were in a position to test our drive for friction to see if it is acceptable for competition performance.

Testing The Drivebase

Friction Test overview

Equipment:
- Drivebase
- Motor – blue (600rpm) cartridge

Method:
- Place a single motor on one of the slots for the motors on **one side**
- Spin motor at max RPM (brain's device control panel) for a minute
- Take reading of motor power consumption
- Take notes on the stability of the power consumption
- Take motor out and allow to cool
- Repeat tests 3 times *for each side*

Results

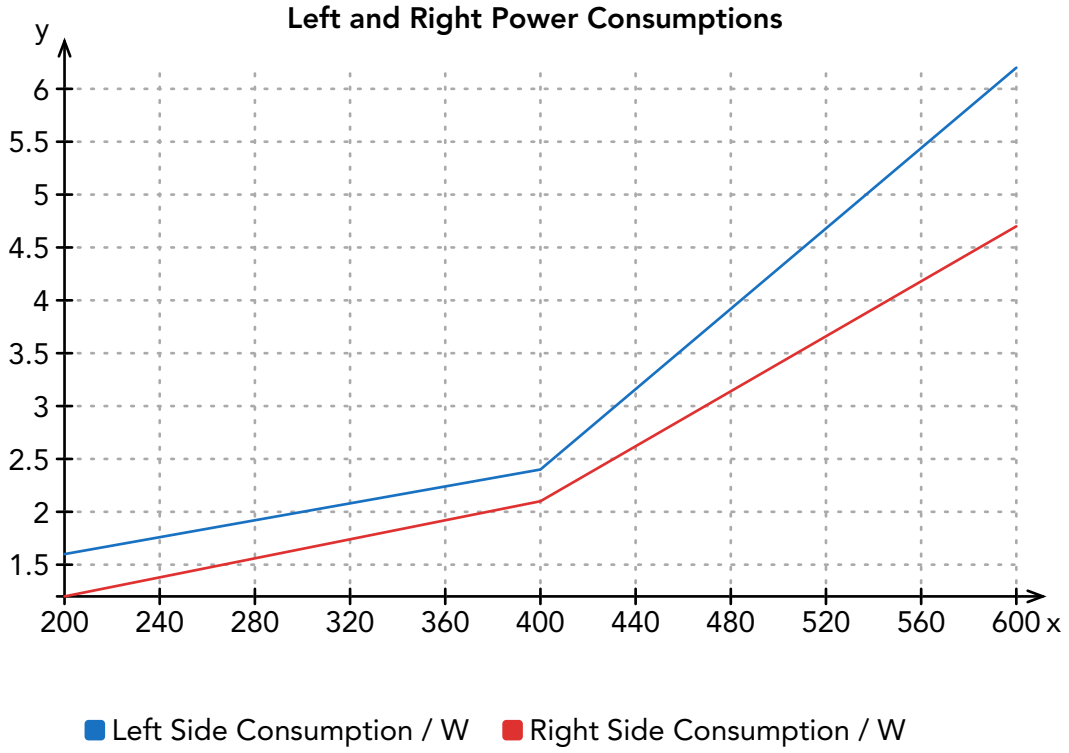
Left Side Power Consumption

Motor Speed / RPM		Power Consumption / W (2 sf)		
	Reading 1	Reading 2	Reading 3	Average
200	1.6	1.7	1.5	1.6
400	2.5	2.0	2.6	2.4
600	6.2	5.6	6.8	6.2

Right Side Power Consumption

Motor Speed / RPM	Power Consumption / W (2sf)			
	Reading 1	Reading 2	Reading 3	Average
200	1.0	1.3	1.3	1.2
400	2.0	1.9	2.5	2.1
600	4.5	4.7	5.0	4.7

Graph of Power Consumption



Conclusion From Results:

- Drivebase has an unaceptable ammount of friction on both sides
- Problems are slightly amplified on the left side of the drive

Possible Reasons For Suboptimal Results

- Not enough slop (tolerance on the wheels)
- Gear to wheel modules over tightened causing warping in the gears
- Bent low strength shaft in motors

- Driving gears contacting wheels
- Inconsistent spacing

Incorporating these results

In our first build through of the drive, we did not take the greatest care and precision to build it. The drive suffered from bowing - meaning the drive was wider in the middle than on the ends causing the width to be between 4-4.7 hole wide. This meant we could not properly brace the drive due to the holes not lining up. This inconsistency in build quality likely contributed to our increased drive friction. We decided the best course of action would be to scrap it and start over, as we would likely use this drivetrain for multiple design cycles and starting with a poor drive would only lead to **issues snowballing**. Below is how we rebuilt our drivetrain from scratch.

> **Note**
>
> We ended up keeping this drivetrain for V1 through V3 of our design cycles. We ended up changing our V4 drivetrain as it was a parallel rebuild and we were using different wheel sizes for reasons discussed in that EDP cycle.

What is a Programming 'Approach'

When tackling any project, thorough planning and thought is required to allow the team to effectively solve it. Defining a standard approach to certain aspects of the challenge can allow certain beneficial procedures to become instinctual, therefore allowing the team to become more efficient.
This is especially prominent when looking at programming, code on large projects can become completely disorganised and hard to read – which is particularly disastrous if you are in a team (where some or most of the members might not intuitively understand the programming language itself).

Common Approaches To Programming

Some **programming best practices** which we wish to implement in our code of conduct are:

- Choosing a suitable language
- Consistent variable naming conventions
- Usage of subprograms or classes
- Version control or project management
- Program structure
- Safe and secure programming

TL;DR

All code written for robotics should follow a standardised set of principals, which are designed to enforce the best practises outlined above. By following the principals we design we should improve the following (ranked in order of importance: high to lower):

1. Team/project management
2. Team communication
3. Readability of code
4. Debugging efficiency
5. Enjoyment of writing code
6. Programming speed

Brainstorming Approaches

Here we will define what approaches we will use when coding, aiming to strengthen team co-ordination, programming efficiency and debugging speed.

Language and Software

One of the most important decisions to make is the programming language and its accompanying software and libraries. For robotics, the two most common languages are C++, Python and increasingly Rust, which is making an appearance with *vexide*[1], but the software used alongside it is also very important.

Python with VEXCode V5 or RoboMesh Studio:

- The only current way to code with python is using VEXCode V5 or RoboMesh Studio

Pros	Cons
• Python is exceptionally easy to read and write thanks to its variable type declaration and readable syntax • Devices can be set up using the built in GUIs	• Using GUI for devices can be restricting • There is no allowances for *any* libraries, something that python is known for • Multiple files are not supported, therefore less orginisation through program structure is possible • No choice in editor, therefore no choice for extensions, themes, formatting etc. • Fairly limited device API, less control over devices

Some example python pseudo code for tank drive (2 stick control) looks like this:

```py
import vex
#Function to instantiate new controller object
control = vex.controller()
#Function that returns all motors in a list
motors = vex.GetAllMotors()

def DriverControl(self, controller):
  #function that returns the left joystick inputs
  movementLeft = control.ReturnLeft()
  #function that returns the right joystick inputs
  movementRight = control.ReturnRight()
  for i in range(len(motors)):
    if i %% 2 == 0:
      motors[i].velocity = movementLeft
    else:
      motors[i].velocity = movementRight
```

[1] An open source Rust runtime for v5 robots

C++ with VEXCode or VEXCode Pro

- VEX offers an alternative to python with C++ in the IDEs VEXCode and VEXCode Pro
- VEXCode Pro allows users to use multiple files, along with header files and other C++ functionalities.

Pros	Cons
• C++ offers much more raw functionality as it is a lower level language[2] • GUIs are still available to configure devices	• Still no access to custom libraries • Rigid IDE, no access to extensions to improve workflow • Fairly limited device API, less control over devices • Cannot declare devices both in code and with GUI, must choose one or the other

Some example C++ code for tank drive (2 stick) may look like this:

```cpp
using vex;
```

C/C++ with PROS

- *PROS* is an open source development environment for VEX founded by Purdue University
- Allows users to write code in C++ or C (C++ is generally preferred)
- Integrated within existing IDEs, e.g. VSCode or Atom

Pros	Cons

[2]A language that is closer to manipulating raw memory, giving users more control over memory

- Allows full C/C++ functionality
- Multiple files supported, including header files
- Full template and library functionality
- Hot/Cold linking: only changed code is uploaded, allowing for fast uploads even wirelessly
- Built into existing IDEs, so extensions, formatting themes etc. is all supported and controlled by user
- Significantly improved device API, allowing for fine control over all devices/ components, including serial inputs/ outputs and direct control over radio
- Easy to get started, but with in depth capabilities for niche applications
- Huge ammounts of documentation e.g. *Purdue SIGBots wiki* or *API docs*

- Can be harder to understand concepts

Some example PROS code for tank drive (2 stick) may look like this:

```cpp
using vex;
using pros;
```

Rust with Vexide

- *Vexide* is an open source 'no-std'[3] Rust runtime for vex V5
- It is a successor to *pros-rs* , which binds Rust code to the PROS API.
- Allows users to code using Rust, while supplying a CLI to manage projects/interact with devices
- Vexide will be included in a family of vex based applications, such as *vex-v5-qemu* , a CPU level simulation for PROS and Vexide code (this also includes node-based GUI for device configuration).

> **⚠ Warning**
>
> Vexide is still considered experimental, it has a small base of contributors that are working to make it more and more usable.

Pros	Cons

[3]'no-std' is a type of package that limits the use of standard libraries. The V5 brain runs without an OS, meaning std libraries are impossible to use.

- Rust is a language designed around memory safetly, including things like variable 'ownership' to avoid memory leaks.
- Vexide will eventually work seemlessly with a range of other projects developed by the vexide team.

- Rust is not an easy or intuitive language to learn – we have very limited experience with Rust
- Because of it being so new, vexide does not have a stable base of users – meaning less documentation and support
- No-std means basic mathematical functions are not accessible[4]
- CLI is still limited especially when compared to PROS

Variable Naming

Naming conventions are very useful when writing *and* reading code. They can make long, complicated names easy to read; and additionally can help clarify the intent, context and scope of a variable in a program.

Variable requirements:

Most languages (C++ included) require variables to adhere to certain rules:

- Must not start with a digit
- Only alphanumeric values or underscores
- No spaces/whitespaces
- No isolated keywords (e.g. 'if', 'for', 'import' etc.)[5]

Common types of variable naming:

camelCase:

Explanation:

- Variable names start with lowercase
- All new words within the variables start with an uppercase

Example:

```cpp
int dateNow() {
  // Get date
}
bool thisIsAVariable = true;
int currentDate = dateNow();
```

Pros	Cons

[4]There are ways around this
[5]Dependant on language

- Easy to understand multiword variables
- Some programs recognise camelCase, allowing them to display variables with whitespaces
- Satisfying variable 'shape'

- Variable names can become long
- Some words can look confusing e.g 'A' in 'thisIsAVariable' ('A' can be hard to see)

snake_case

Explanation:

- Using underlining to represent whitespace
- Words typically start with lowercase

Example:

```cpp
int date_now() {
  // Get date
}
bool this_is_a_variable = true;
int current_date = date_now();
```

Pros/Cons:

Pros

- Very easy to understand understand multiword variables
- Very easy to see where whitespace is supposed to be

Cons

- Variable names can get very long
- Somewhat difficult to program with due to frequent use of '_'

> **Note**
>
> The difficulty from frequently using '_' can be circumvented by using a program such as Auto HotKey to rebind '_' to something such as "Shift" + "Space".
> However, this work around may not be worth it for the express purpose of making a naming convention easier.

Boolean 'is' naming

Explanation:

- Start all booleans with 'is'
- Often times subprograms that return boolean values start with 'get' ('getIs…')

Example:

```cpp
// (Using camelCase)
bool getIsSaturday() {
  // is it a saturday?
```

```
}
bool isSaturday = getIsSaturday();
```

Pros/Cons:

Pros	Cons
• Easy to differentiate between regular procedures and functions that return a boolean value • Works best with camelCase but can also work with other variable naming conventions	• Variable names can get very long • Doesn't help identify between procedures and functions that return other data types

> **Note**
>
> Procedures are defined as subprograms, which don't return **any** value, whilst functions are defined as subprograms, that return a value of a **specified** data type.[6]

Pronouncable names

Explanation:

- Using abreviated words that are pronunciable and easy to make sense of

Example:

```cpp
int getCurrDate() {
  // Get date
}
bool thisIsAVar = true; // variable = var
int currDate = getCurrDate(); // current = curr
```

Pros/Cons:

Pros	Cons
• Drastically shortens variable names	• Not all abbreviations will make sense to everyone • Not using standard english can make understanding code harder • Text autocompletion in IDEs means that long names aren't a problem to type.

[6] Languages such as Python do not require such precision in subprogram declaration mitigating the difference between functions and procedures. However, most other languages require procedures to be declared with the **void** keyword to specify they don't return a value.

Unit classification

Explanation:

- Suffixing all variable names (where applicable) with a unit (e.g. rpm, lbs, kgs etc.)
- Using an underscore to seperate units
- Best used with snake_case

Example:

```cpp
int getMotorSpeed() { // Don't need unit classification for subprograms
  // Get RPM
}
int motorSpeed_rpm = getMotorSpeed(); // suffixed with '_rpm'
```

Pros/Cons:

Pros	Cons
• Units are always known • Conversions can be easier because input/ output is documented in the name	• Takes longer to document all variables' types • Increases variable length[7]

Using Subprograms and Classes

- Subprograms are smaller blocks of code that can be run anywhere in user code
- Classes are structures that allow for variables to be 'owned' and can drastically help organisation[8]

Pros	Cons
• Organised and readable code • Similar code can be run multiple times without being repeated • Classes allow for even further organisation • Classes can help mitigate developer mistakes	• Classes can take time to write • Variables defined in different files can lead to null pointers during initialising[9]

Version Control

An equally important issue for programming as a discipline is Version Control. This is important because it allows multiple people to work in parallel on the project from anywhere to develop one feature or several and then merge changes into one main branch of the project. In addition, Version Control softwares allow developers to mess with experimental changes without the fear of ruining the project, and it also acts as safety net incase a changeset that enters the main branch ends up breaking the project, since it can easily be

[7] It doesn't increase it by much and also text autocompletion is still a thing
[8] We may do a deep dive on classes at a later point
[9] Basically, the C++ compiler does not have a specified order for intialising variables in different files, meaning if 1 variable depends on an unititialised variable (from another file) it can throw a memory error

rolled back to a stable build. Another advantage of Version Control is that it allows for new and old code to compared for performance over whatever parameters we test - allowing us to easily prove if our new solution is an improvement.

Examples of Version Control Software

There are many different Version Control Software. For this logbook, we are using GitHub - which is built off Git, since it has direct support for Typst and Notebookinator. Other examples include:

- Beanstalk
- PerForce
- Apache Subversion
- Mercurial

Program Structure

As well as using subprograms and classes for organisation, the usage of libraries and splitting the code up into different files helps keep the software for the robot structured and helps prevent accidental changes when working on different parts of the code. Our experience from last year taught us that its best practise to have the code for driver control, match autonomous and autonomous skills on seperate files. In last years game OU, this principal was taken further since we had different files controlling our left and right side autonomous routes for qualification and elimination matches.

> **⚠ Warning**
>
> Global variables have to be initialised in 1 file to avoid Null Pointer Exceptions since the C++ compiler doesn't specify initialisation order.

Safe Programming

When working in medium level languages such as C++, the necessity for optimising your code and ensuring you handle computer memory and pointers properly increases. Failing to do so effectively, will cause crashes and problems on your machine and at a large scale can even throw the world into *disarray* .
If we use C++ again this year, we must make sure to initialize variables in the right order and ensure we handle memories and pointers properly. General things we can do ensure this are: using smart pointers, using tools such as static analyser; and finally using STL containers.

Smart Pointers

These help reduce the amount of manual memory management. The Microsoft *documentation* states that they are crucial to the RAII or Resource Acquisition Is Initialization programming idiom. The basic premise of RAII is to ensure finite resources are not wasted and so must control their usage and destroyed once its no longer useful - ie when a variable goes out of scope. The usage of Smart Pointers greatly reduces the chance of bugs and memory

leaks since memory is automatically deallocated when the resource is no longer used. Below is a comparison of a raw pointer vs smart pointers.

```cpp
void UseRawPointer()
{
    // Using a raw pointer -- not recommended.
    Song* pSong = new Song(L"Nothing on You", L"Bruno Mars");

    // Use pSong...

    // Don't forget to delete!
    delete pSong;
}

void UseSmartPointer()
{
    // Declare a smart pointer on stack and pass it the raw pointer.
    unique_ptr<Song> song2(new Song(L"Nothing on You", L"Bruno Mars"));

    // Use song2...
    wstring s = song2->duration_;
    //...

} // song2 is deleted automatically here.
```

> **Note**
>
> For the majority of C++ programming, smart pointers aren't necessary to manage especially within the context of VEX programming.

A Quantitative Approach:

While we typically use quantitative methods for deciding aspects of our robot, such as the drivetrain (for example using decision matrices); we will avoid using such methods when deciding our programming approach. While we believe taking care, when picking a programming approach is *very* important, much of it is still personal preference and can be decided by simply evaluating what fits us best.

Deciding Our Programming Approach

Here we will display the choices we made on the different aspects of the programming approach and some explanation as to why.

Language and Software

Final Decision

We ultimately decided to code in C++ using PROS

There were many factors involved with this, but experience played a big part in us choosing this: we used Python with VEXCode for the majority of last year, but we found it was very limiting, with the lack of multiple file support and library support. We also used PROS/C++ last year for nationals and worlds, and we found it to be very reliable, even if we didn't utilise the libraries available to the fullest.
Our team leader, Daniel Dew, did a significant amount of research on Rust with vexide, talking to the main developers about the state of the project. From his conversations, we found that while it seemed like a very interesting project, there was not currently enough surrounding support and documentation necessary to justify switching to the less familiar language: Rust.

Variable Naming

Final Decision

We decided to use the following variable naming conventions:
- camelCase
- 'is' boolean variable naming
- Pronounceable names

Variable naming is mostly inconsequential, so we chose what we were most familiar with. It is important to stick with this, as collaborative programming is much easier when the conventions are stuck to.
Arguments can be made to say that stricter rules must be followed, for example this *video* states why long variable names dont matter, and that unit classification is important. However, in VEX, code changes are often made in a hurry, and its easier to focus on a few instinctual rules than to have strict ones that force the programmer to pour over the code after rushed

changes are made to fix formatting – we are also unlikely to go above 1-2 people making changes to the code, so punctuality matters less.

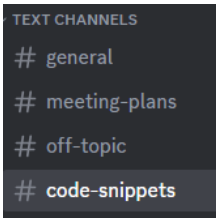
Version Control

As programmers get more experienced, it becomes more and more apparent that version control is a **must have**; as projects become exponentially more complex, the appearance of bugs in code becomes a question of when rather than if. Therefore, the ability to revert code into a stable state is essential.

> **Final Decision**
>
> We decided to use the most common form of version control: Git

Git (using *GitHub*) allows us to easily collaborate and use code accross multiple devices, with branching functionality and the ability to manage multiple projects as an organisation[1].

> **Note**
>
> Small changes that we need to implement on another device can be shared in a discord channel, to avoid cluttering the version control.



Small changes are shared in our "code snippets" channel.

Other approaches:

These are approaches that don't require decisions (you either use them or you don't)

- Subprograms and classes will be used where applicable[2]
- We will prioritise using multiple files to aid with organisation
- We will adhere to general safe programming rules, especially avoiding unititialised variables, as the compiler doesn't catch them and these issues can be extremely difficult to debug.

Sticking to the Rules

With all these 'rules' it may seem like it will be impossible to follow these in practise. However, we can implement procedures to help aid this process. We have taken inspiration from how many companies maintain their huge codebases.

[1]Our organisation: *https://github.com/29457A-SNowflakes*
[2]Classes, in the context of VEX can become more of a nuisance than an aid, so they must be used sparingly

Example

All of Meta's code (including Instagram, Facebook, Meta VR etc.) is all kept on the same Git codebase, meaning Zuckerburg's original Facebook code is available to *all* meta developers

- Before commiting to the Git, we will make sure that all code follows the formatting rules
- If code changes have to be made in a rush – for example during a competition – commits to the Git will be flagged as bad code (often using the 💩 emoji[3]), and will be formatted at a later date.

[3]This is actually an industry standard flag for bad code

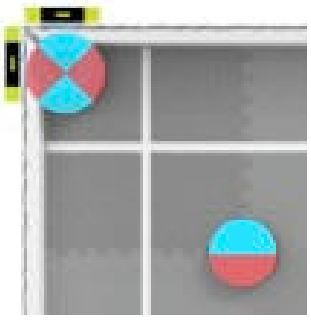
> **Note**
> This notebook entry was written by our head of tactics and will communicate in the first person - rather than the typical third person style of these entries due to the nature of this topic being almost entirely Thomas's domain.

My role and the manual

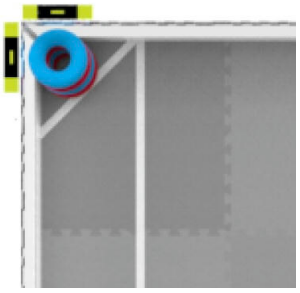
As head of tactics, we decided that it should be my responsiblity to know absolutely everything in the game manual. Having extensive knowledge about how the game is scored and what referees are watching for helps me to work out what the best strategy is in certain situations in order to get us the most points and avoid recieving violations.

Over the summer, I spent a lot of my time memorising the manual, so I would be ready for our first competition, and I was excited to see what would change in the September 3rd update, which was the first major update of the manual since its release. These next few pages will summarise the key things that I took from this update, and the impact that I believe they will have on the game subsequently.

Field tape layout and SC5 change



Field before update [1]



Field after update [1]

This is one of the most major changes in this manual update, as it not only greatly reduces the amount of space for the mobile goals, but it also makes it much easier to tell, which mobile goal is considered scored. It is also relevant to note that - according to SC5 - for a mobile goal to placed, there is now 3 criteria:

1. The mobile goal is contacting the floor/tape line
2. A part of the flexible top is above the field perimeter
3. The mobile goal is breaking the plane of the corner

I believe part 2 and 3 are especially significant, as in the previous version, the stake didn't need to be 'upright', and this change makes it so its harder to score. As if the stake is tipped over by the opposing alliance, it will not get the corner modifier.

Part 3 is also significant as the rule used to be: the base of the stake must contact the floor of the corner, but now the edge of the base can just overhang, making it slightly easier for the mobile goal to be considered as placed.

SC3 rewrite

In my opinion, this is the single biggest strategic change to the game manual, and will greatly impact many future matches. SC3 used to have 4 conditions:

1. Rings cannot touch the floor
2. Rings cannot be touching the same colour robot
3. Rings must be encirling the stake
4. Rings cannot exceed the maximum number of rings for the stake

However, the new SC3 rule overhauled these conditons, and only made the last 2 relevant. This is **game changing** because if a ring is 'on' a stake, it is now pretty much guaranteed to be scored. Whereas before, teams had to think about whether their robot was touching the rings or if the stake was toppled over, but now these 2 conditions are irrelevant. This will make scoring much easier, and I can imagine we will see many more points given to both teams in future games.

AWP for NATS/Worlds

In the September update, we finally got the conditions for the AWP for Worlds and Nationals competitions. This year, we are determined to try and have a robot which can solo Auton for an AWP, as I believe that AWP's are massively underrated, as they are worth 1/2 the points that winning a match is, and, if done well, can consistently be done.

I was expecting that the AWP would have maybe 1 more ring scored, something just slightly harder than the "normal" AWP, but how wrong I was. Now, the AWP forces you to have a ring on the alliance wall stake, and although it seems simple, I cannot stress enough just how much harder this will be. It will completely change the route of the robot and also means that the robot will have to pick up an extra ring. Although this seems easy, this forces the robot to get a ring from a pile stacked with the other alliances coloured rings, and for some robots, it will be difficult to seperate the blue from the red rings, potentially helping the opposing alliance with their autonomous bonus, making your alliance at a disadvantage before the driver controlled period even begins.

Although the AWP is much harder to obtain, I believe this is a positive change, as, from my experience, in previous games, the AWP was sometimes too easy to get, and therfore the tournaments relied less on the actual driver controlled section and teams could easily get Win Points mainly through AWP. However, if a team can perfect their autonomous, they could consistently get the AWP, but this will require a lot of work and overall I believe that this is a good change.

SG4 rewrite

Although many teams believe this rule is less significant, and some may glance over it completely, I once again believe this is an incredibly important rewrite. SG4 used to state that if a ring left the field, it would result in a minor violation for the team, and 3 rings would cause a major violation, and therefore an automatic disqualification, and this would apply for any future games too.

This would stop teams from trying to descore even-points neutral wall stakes, as there would be the fear of pushing the rings out of the field instead of back onto the floor tiles, and therefore would incentivise teams to put rings on a wall stake that already has some of the other alliances rings as they knew there would be little risk of another team trying to descore, as they would be worried that they would descore their own rings, and possibly get a violation, as well as losing some of their own points.

However, due to the new rule only applying to the enemy alliance's rings, this means teams will feel more comfortable descoring their own rings, as there is less risk of getting a violation. I believe this rule change makes more sense, as teams shouldn't be punished for losing their own points.

The Need For a Mobile Goal Mechanism

Scoring in High Stakes is a complex engineering challenge. This is because, unlike in last years game[1], it requires a dedicated and precise mechanism in order to put rings on the stakes - both mobile and static. Additionally, from analysis of the rules and games played [insert relevant footage] we have realised the importance of being able to manipulate and hold mogos.

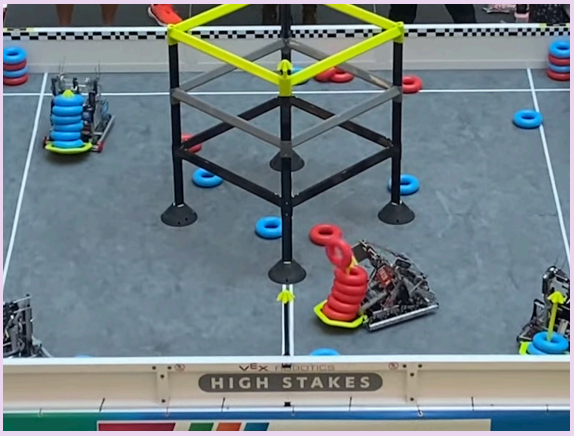
Therefore we need to design a subsystem that can:

- Grab and release mogos
- Securely hold grabbed mogos

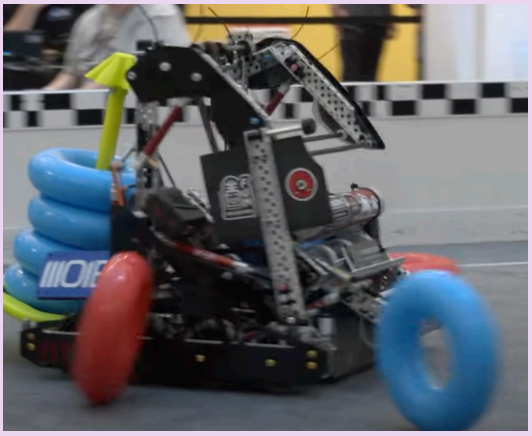
Example

Further Motivation

One case study for the need of mogo manipulation is the early season 'Mall of America' (MOA) signature event. Throughout this competition, the manipulation of filled or semi-filled mogos was key to winning a match – in most, if not all, it was actually the deciding factor. During our analysis of the key games during MOA, and our analysis and familiarisation of the manual, it became increasingly obvious that leaving this aspect of High Stakes out of our design would be a bad decision.



A snapshot from a video taken during the MOA finals match 1 [10], showing all robots manipulating mogos to their advantage.

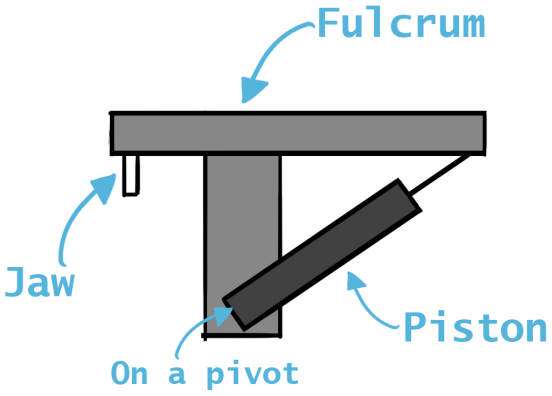


One of the Tournament Champions from MOA, 11101B Barcbots, using a mogo & clamp. (Snapshot from Pits & Parts showcase [11]).

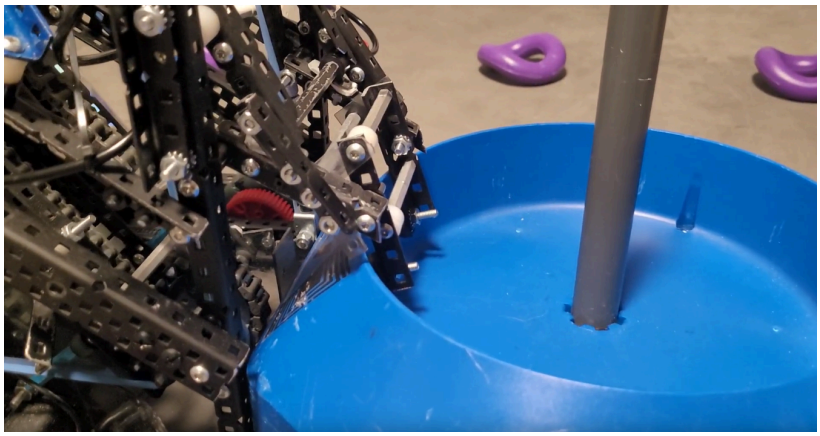
[1] In OU even basic push bots could see relative success at scoring since triballs could be simply pushed to score.

Piston based clamping

The two most common ways to clamp things in VEX is by either using pistons or motors. We will start by looking at piston based clamping. In general, piston based clamping works using a lever applying effort to one side to support a load on the other. Below is a basic concept sketched in photoshop.



Looking at clamp designs for mobile goals, we came across two designs, which shared various similarities. The first design comes from *@6047OSSemicolon* [12] for the game Tipping Point, a game which also features mogos with a similar design. The premise of the design uses a piece of laser cut polycarb, two pistons and two rubber tips to pin the goal to the side of the bot. Notably, HS mogos don't have wide walls and are instead much flatter. However, the concept of this design could be reworked for a High Stakes bot.

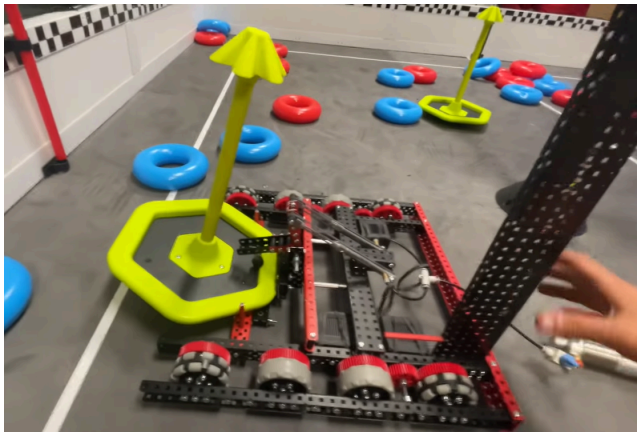


60470S *mogo mech [12]*

An example of a mech from this years game is this prototype by 23851A [13], which uses two pieces of C-Channel and some standoffs to align and angle the mogo. The actual clamp is made up of two cut C-Channels attached together with a gusset. Similarly it uses two pistons, which when extended causes it to pivot and clamp down on the mogo base.

Brainstorming our V1 Mogo Mech

2024/09/02



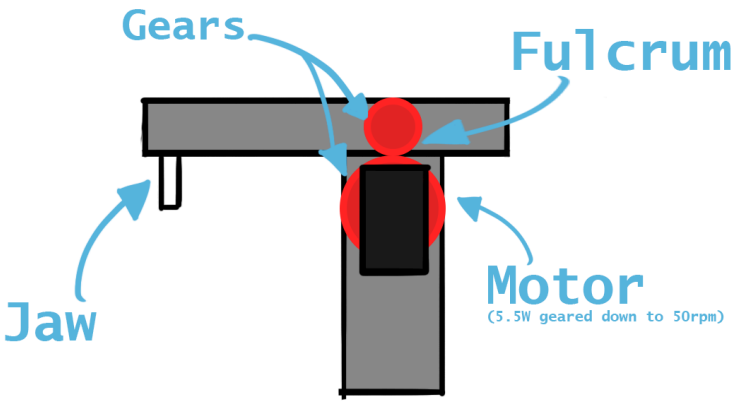
23851A *mogo mech [13]*

Here's a list of advantages and disadvantages for a piston based solution.

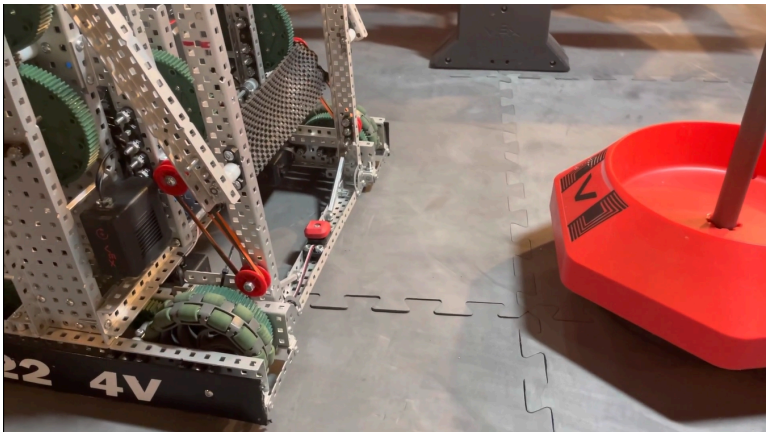
Pros	Cons
• Fast extention time • Doesn't contribute to power budget • No burn out • Better gripping • Lighter mechanism	• Has much more limited actuations than motor based solution • Pneumatic tanks have to be filled before each match

Motor based clamping

An alternative to piston clamping is using motors as a power source. Motor based clamps generally work by using motors as a pivot. Below is a sketched example of how such a thing could work.



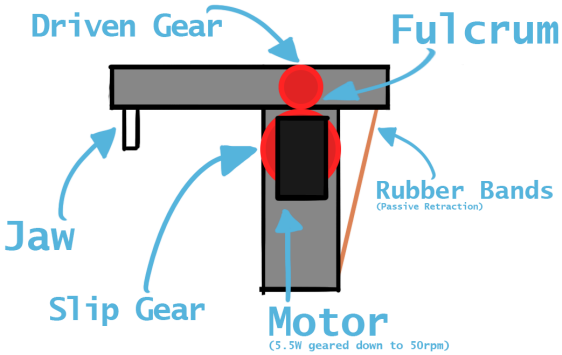
Looking around for inspiration, the best design we came across was made by *@22204V*[14] for Tipping Point, which uses a single motor with rubber bands to tension it, giving the design passive retraction and active extension. However, this design acts more like a scoop than a clamp relying on supporting the clamp from below and the sides rather than pinning it to the side.



Example from the video . [14]

Here's a list of advantages and disadvantages for a motor based solution.

Pros	Cons
• Avoids the sometimes volatile pneumatics • More conistent actuation, independant from previous actuations	• Takes 5.5/11W from 88W limited • Typically weaker than pneumatics • Typically slower actuation than pistons

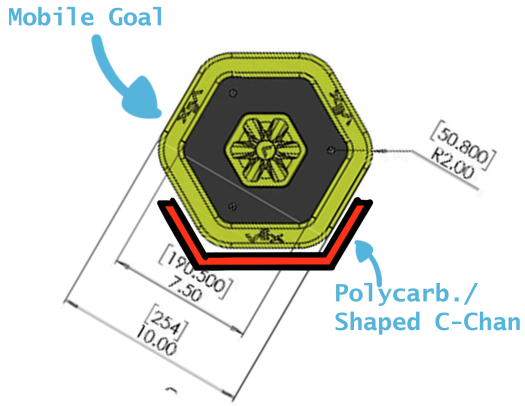


A more experimental concept, which could use slip gears and rubber bands for active grabbing and passive retraction, based on the ideas from the example above.

Passive mechanism

Another alternative to clamping is creating a mechanism to align the mogo to plough it. The benefits of this solution is that it is less complex to implement meaning less chance of failure,

in addition it doesn't use pistons or motors so it doesn't come with either of their drawbacks. Below is an example of how that could look.



Here's a list of pros and cons for this solution.

Pros	Cons
• Doesn't contribute to power budget • Doesn't use up PSI or need pneumatic tanks • Simple	• No gripping power • Mogos can be lost on turns • Mogos can be stolen by opposing bots • Little control on positioning for ring mech • Restrictive: can't go full speed or reverse with mogo

Deciding on our V1 Mogo Mech 2024/09/04

Deciding on the mechanism

To decide on a mogo mech architype we will compare each of the qualities of each design.

- Grip
- Actuation amount
- Complexity
- Manoeuvrability
- Activation speed
- Weight

And additionally the requirements of each design will be considered seperately.

	Grip	Actuation amount	Ease of build	Manoeuvrabil	Activation speed	Weight	Total
Piston	5	3	4	5	4	4	25
Motor	3	4	3	4	3	3	20
Plough	1	5	5	1	5	5	22

Piston requirements:

- Pneumatic tank

Motor requirements:

- 5.5W/11W from power budget

Plough requirements:

- Polycarb or standard metal

Ultimately we have decided to discard the plough as a solution since it drastically reduces our manoeuvrability since we can not drive or turn at high speeds with a mogo or reverse while trying to control it. In addition, not holding the mogo in a fixed position and angle means that

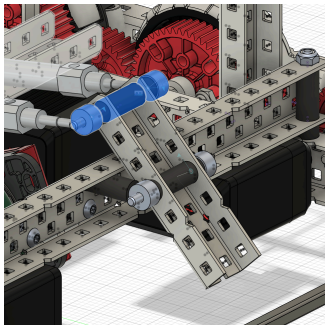
it is much more difficult to score rings on stakes. These factors mean that choosing a plough design will greatly limit our match performance and jeopardise our chances of success. However, the usage of a cut polycarb piece for alignment still seems like a useful mechanism to help us manipulate the mogo and therefore more accurately score rings on mogos.

Final Decision

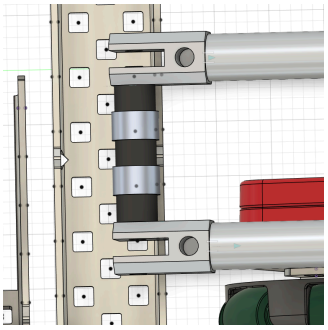
Between a piston based solution and a motor based solution, we have decided that a piston based solution is the superior choice. This is because in general pistons are able to perform the task better in almost everyway, with the greatest draw of motors being more actuations. However, this is still a non issue since if we have two tanks with our two piston design then we will have at least 20 actuations per game, which from game analysis [insert relevant page/web link here] is much more than needed in match conditions. Therefore, we will iterate on a piston based solution.

CADing The Mogo Mech

As with any part of our robot we first like to start any build process by modelling the part on CAD, the benefits of this is that it provides an exact formulaic instruction to follow that theoretically works in the way we want it to, this allows us to have complex builds with parts in hard to reach places that we can prepare in advance, the reason this is so good is that it decreases rebuilds and failiures, making sure that we maximise our time and don't have to keep taking everything apart because a bearing is out of place



Completed Mogo CAD side view



Completed Mogo CAD

Here we can see the completed CAD model of our mogo clamp which we will now build out of physical parts,

The build process:

As we have now finished with the CAD part of the construction process we can now create a parts list that can be used to build the clamp

- Shaft Collar (6X)
- 0.25in Spacer (4X)
- 0.5in Spacer (6X)
- 0.125in Spacer (4X)
- 2.5in Screw (3X)
- 1.5in Screw (4X)
- Nylock nut (4X)
- Small Piston (2X)
- 1x2x1x7 C channel

The way this mogo mech works is by using pivots and pistons to keep the goal at a desired angle and so we can carry it round the field. We will now build the clamp and observe results (these finicky mechanisms often require a large amount of trial and error)

Why we built like this:

- This design utilises a pivoting C-Channel which is cut just to the length where the power is far enough away from the pivot that it provides as secure and stable clamp using the least

power but also small enough that it does not add additional weight or unnecessary parts to our robot that may obstruct future mechanisms,
- this C-Channel is powered by 2 small pistons which we use to find a compromise on power, space and weight,

– they are the smallest so provide the least extension. This is not an issue in this case as the clamp does not have to extend far, – they are small so lighter (if marginally) than the other sizes – most importantly they are the smallest so do not have to be mounted at an inconvenient spot on our drivetrain as motors/bearings are often in the way,
- this piston movement pushes one way and clamps the other allowing the goal to tilt in the favoured direction, we used a screw joint mounted to the C-Channel with shaft collars to ensure that the pivot was smooth, allowing it to use as little air as possible

Step 1

In order to build this we first created the piston mount seen in the photo, this required us putting a 2.5in screw through, shaft collars and spacers and each piston in order to secure them together and provide a mounting point for the pistons above the C-Channel attached to the drivetrain (**the shaft collars provide this as you can screw into them at a right angle**)

Step 2

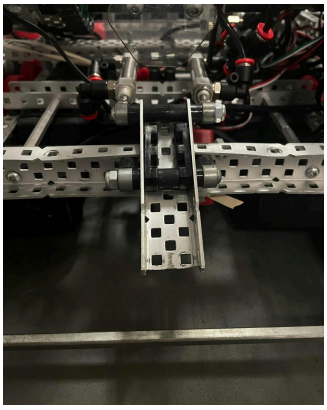
We then constructed the C-Channel which will be used to clamp the mogo into the angle we desire, as seen on the CAD used spacers to make sure that the C-Channel is “boxed” and will not deform under heavy use, here we deviated slightly from the CAD and utilised bearing flats to make sure that the screw is centered when going through the C-Channel, this decreases friction.

Step 3

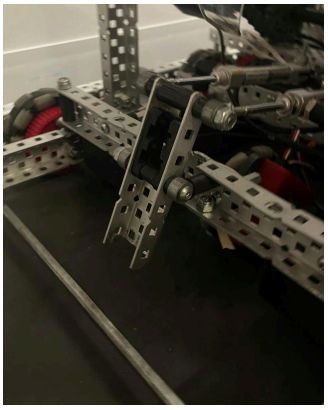
We then used the same method as attaching the pistons to mount the C-Channel on the first pivot, with screws coming out horizontally from the C-Channel we attached them to shaft collars that are attached by a srew running through the C-Channel.

Step 4

Finally we used the same method to attach the pistons on a second pivot at the top of the C-Channel, allowing us to utilise the pushing, clamping power of the pistons on the Mogo.



Here is the mogo clamp that we built using the instructions and parts listed here



Here is it extended as it would be with a mogo inside

? The Need for Autonomous Sensors 2024/08/20

The Problem

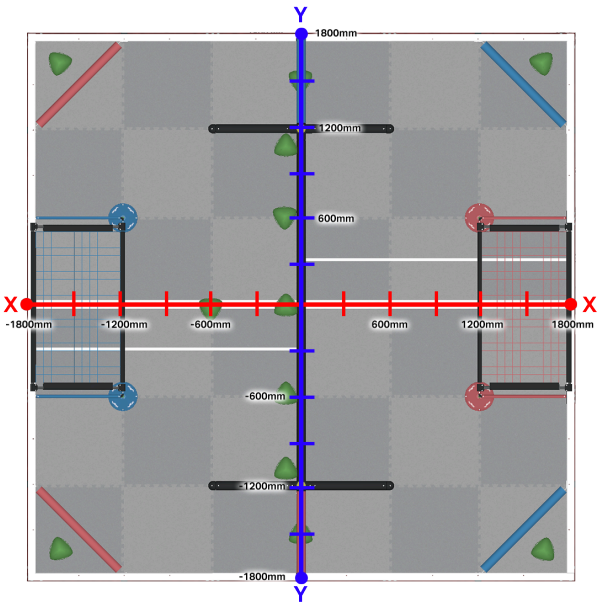
During every match, there is a 15 second autonomous period; there is also a minute-long autonomous skills challenge – which contributes towards our skills score. Ultimately both autonomous challeneges boil down to 1 simple challenge: finding where the robot is.

Finding The Robot

To know where to go, the robot must first know where it is; generally there are a lot of ways to find this – or the robot can simply guess based on rudimentry time inputs.

Where Is This Applicable?

Knowing where the robot is can be useful in a range of scenarios, but of course the main one being during designated autonomous periods (either at the start of a match or during auton skills run). It can also be applicable during driver periods – for example to run macros that require moving the robot.
On the right, there is an image – taken from one of VEX's guides to autonomous [15] from over under – showing the global[1] dimensions of the field, these can be used to program autonomous routines.



Field Dimensions defined uisng the centre as the origion(12'x12') [15]

Other Necessary Aspects

To make full autons, we also need to know the state of various other aspects of the robot – including:

- State of pistons
- State of subsystem motors
- (Sometimes) time elapsed from start

Easy solutions

Some of the aforementioned aspects are very easily handled (and therefore will be ommited from the brainstorm process), more specifically, they can be easily handled within the code with minimal extra knowledge.

[1] aka a birdseye view

Designed by: Daniel Dew
Witnessed by: Jonah Fitchew

State of Pistons & Motors

```cpp
pros::adi::Pneumatics thisPiston; // declare piston using in-built 'Pneumatics' struct
thisPiston.toggle(); // toggle piston extension
bool extended = thisPiston.extended; // access state of piston

pros::Motor thisMotor; // declare motor with pros 'Motor' struct
moveMotor1Sec(&thisMotor) // arbitrary function to move motor for 1 second
float motorPos = thisMotor.get_position_relative(); // access state/position of motor
// Other telemetry functions are included for the motor struct.
```

Elapsed Time

```cpp
float time = pros::millis(); // time in milliseconds from start program
```

Aproaching Sensors

As always, it is best to have a plan on how we are going to approach picking which sensors and methods to use. There are many factors involved, but some are much more important than others; for instance, we are very keen on reliability – and, for early season, we want to focus on high scoring 'support' autons – therefore we can focus our search in favour of reliabilty and (for high scoring) speed. Cost is also a big factor, so we must be careful to do thorough research on all possible solutions to avoid misusing our budget.

Brainstorming Sensors

Ultimately, the combinations that are possible in VEX are almost limiteless – for that reason, we will only brainstorm what we feel is likely to be effective or has been proven to work.

> **Note**
>
> **Odometry**
> Odometry may be mentioned throughout the brainstorm process, it is a method of absolute positioning using encoders and/or IMUs. If we move forward with odmetry, we may cover it in seperate page(s).

Internal Sensors

IMEs

Not to be confused with IMU, an IME stands for 'Internal Motor Encoder', essentially, they measure how many rotations the motor has done; they can be reset/tared to 0, and various different measurements can be pulled from them, such as rotation in degrees, radians or full rotations.

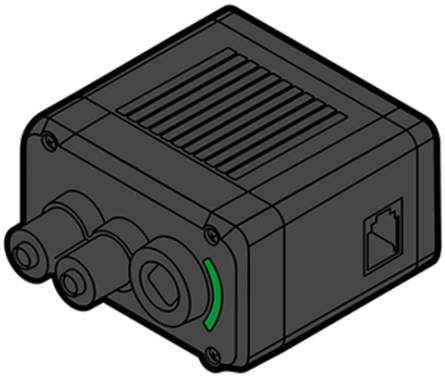


Image of a standard 11W VEX Smart Motor (includes IME) [16]

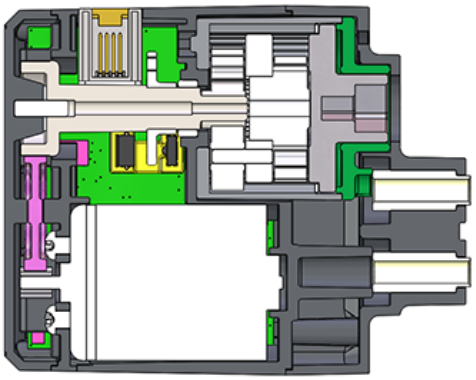


Image of the VEX Smart Motor's (11W) internal structure [16]

IMEs can be used in a range of cases – from finding the state of a motor-based subsystem (see page before), or used in the drivetrain as an input to an odometry program.

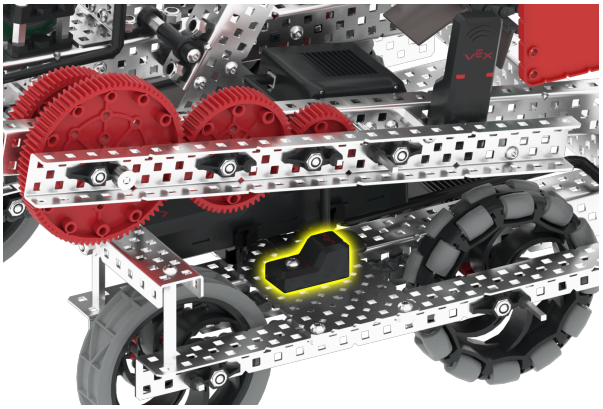
Pros	Cons
• Build into the motors, no extra hardware needed • No extra space needed • Accurate on slow moving systems	• Inaccurate after prolonged use • Encoders attached to powered input: cannot account for wheel slippage, gear skipping or other linkage-based inconsistensies

External Sensors

IMUs

IMUs stand for 'Inertial Measurement Units,' they are widely used sensors that can collect a range of data:

- Gyroscopic measurements (acceleration in 3 axes)
- Heading (relative heading from calibration – clockwise, degrees or radians)



An inertial sensor installed on a robot (OU Striker/Hero Bot) [15].

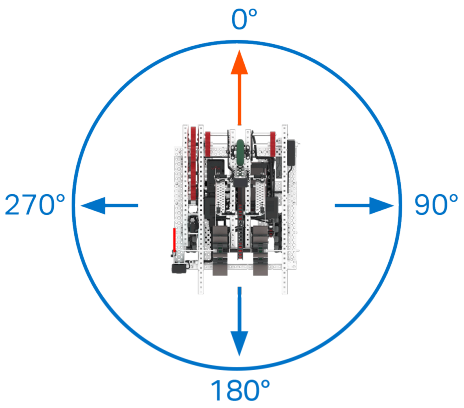


Diagram showing heading directions North, East, South and West in degrees [15].

Ways to use IMUs

- Heading: heading is of course very useful data during autons, they not only allow the robot to 'know' which way it is facing – but can also be used to drastically improve the accuracy of an odometry program.
- INS: 'Inertial Navigation Systems' is another absolute position system that integrates acceleration into velocity, then into position.

Equation

$$A_f(t) = \begin{pmatrix} \cos(-\theta) & \sin(-\theta) \\ -\sin(-\theta) & \cos(-\theta) \end{pmatrix} \cdot A_b(t),$$

$$P_f(t) = \int_0^t \int_0^t A_f(t)$$

Where:

- $A_b(t)$ is the IMU acceleration vector in form $\begin{pmatrix} x \\ y \end{pmatrix}$
- $A_f(t)$ is the global[1] acceleration transformed from $A_b(t)$ in form $\begin{pmatrix} x \\ y \end{pmatrix}$
- $P_f(t)$ is the current field orientated position of the robot (integrated from $A_f(t)$)

The process of using the IMUs data in IMU odometry (INS)

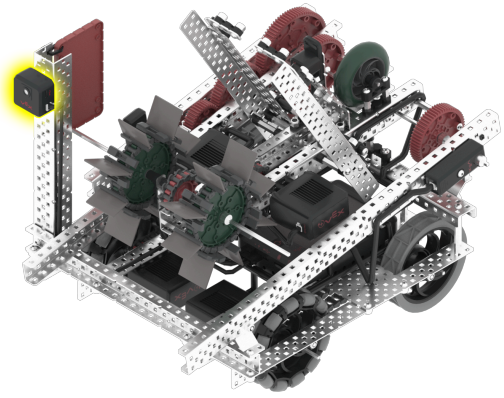
[1]Field orientated

Pros	Cons
• Very accurate heading • Consistent and reliable • Very useful in odometry	• Extremely noisy gyroscopic measurements (INS unusable)

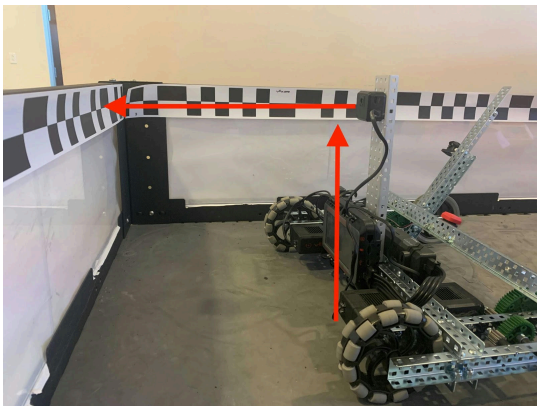
GPS Sensors

GPS stands for Game Positioning System; the VEX GPS Sensor (not to be confused with sattelite-based GPS) is a small box that has a camera, the camera views a QR Code-like pattern on the field perimeter, and computes the global position based on the pattern, this data is streamed to the brain as coordinates with the centre of the field as the origin (see image on page 176).

> **⚠ Warning**
>
> GPS Must maintain a constant line of sight with the field perimeter, and they must be at the correct height.



A GPS Sensor installed onto the OU Striker/ Hero Bot [15]

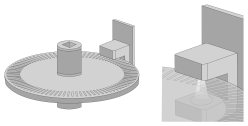


An example of a GPS Sensor with correct positioning [17]

Pros	Cons
• Easy to use (in code) • Somewhat accurate position	• Slow update rate (25Hz in theory – much slower in practice [18]) • Innacturate Heading measurements in practice [19]

Optical Shaft Encoders

Optical Shaft Encoders (OSEs) are sensors that use 2 ADI ports to stream the current position of a low stregth shaft (inserted into hole in sensor). They have a resolution of 1 degree per tick[2].



The disc contained within the OSE for optically measuring rotation [20]

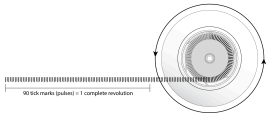
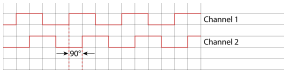
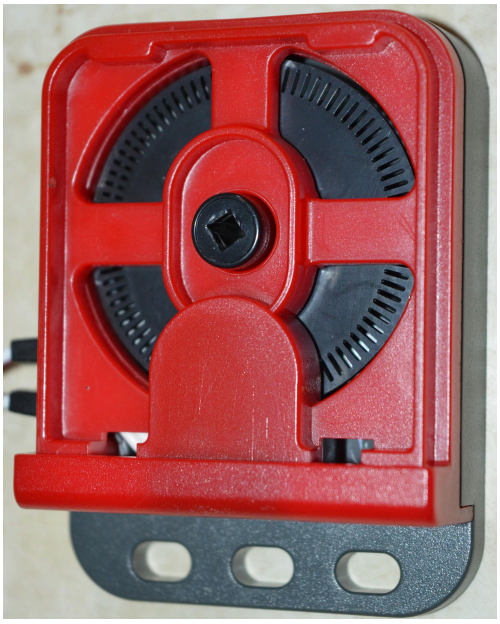


Diagram explaining how ticks are converted to rotations [20]



The signal phase produced from the ticks[3] [20]



The inside of a OSE [20]

> **⚠ Warning**
>
> OSEs are now discontinued, in favour of the Rotation Sensors, we however bought some during OU season.

Using OSEs

State of Subsystems:

Some subsystems that rely on rotation can be measured using an OSE, either to replace the sometimes inconsistent IME, or to measure rotation on a piston based rotational mechanism.

Pros	Cons

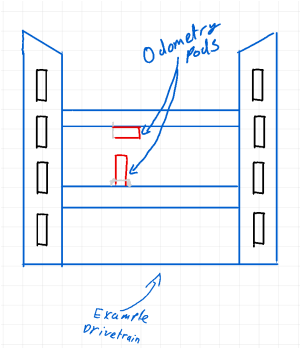
[2]'Ticks' are digital pulses sent when the shaft moves a set ammount
[3]Phase indicates direction

- Can measure rotation on piston based subsystem
- Can be used after gearing to counteract gear skipping affects

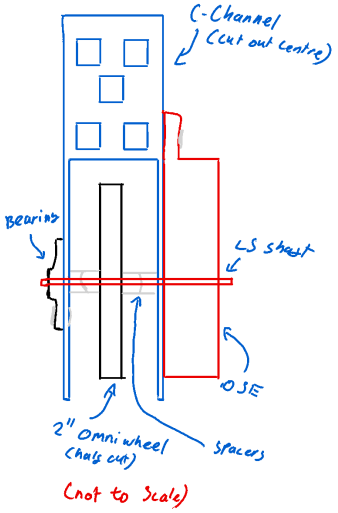
- IMEs typically suffice for motor based application
- Piston based mechanism rarely need to be measured, as they can't be controlled

Odomety Pods

Odometry Pods are essentially a wheel attached to an OSE, they can transform rotations into distance travelled. When they are attached in a certain orientation, they can measure distance travelled in either direction[4].



Example placement of Odom Pods (perpendicular)



A labeled diagram showing a space efficient odometry pod (requires half-cut omni wheels)

fx Equation

Encoder rotation to distance travelled:

$$D = \frac{\Delta\,\Omega \cdot C}{2}$$

Where:

- D: Distance travelled
- $\Delta\,\Omega$: Change in rotation of encoder in radians
- C: Circumference of wheel

[4]Works significantly better with an IMU

> **Note**
>
> While other orientations of odom pods exists (using 3 of them), we are not looking at them due to their decreased performance gain (and the fact we would have to buy more OSEs, which is impossible)

> **Note**
>
> To find absolute position of robot using odometry pods, an odometry algorithm needs to be used – the maths is out of the scope of this research and can be viewed *here* [3].

Pros

- Very versatile
- Reliable
- Very low resolution (1 degree, ~0.023" on 2.75" wheels)
- Pre-built libraries allow for fast programming with odom pods

Cons

- Difficult to program without libraries
- Can require half-cutting omni wheels (lower strength)
- Uses 2 ADI ports

Rotation Sensors

Rotation Sensors are practically identical to OSEs, with the main difference being size and connection type (smaller and uses smart port)

Image of rotation sensor from the VEX Store [21]

They can be used in the same configuration and context as OSEs, and are often more favourable for odom pods due to their reduced size.

Pros	Cons
• Smaller than OSEs • Lower resolution (0.088 degrees) than OSEs	• We don't have them • They are £43.19 each (we would need at least 2)

Positioning Takeaways

From our research, these sensors can be used in the ways depicted to find position, either relatively from starting point or absolutely from the centre of the field.

Complimentry Sensors

When implementing autonomous routines, there are a range of complimentry sensors that can be used alongside the positioning sensors to further improve the complexity or reliabilty of the routines.

Complementary Sensors

When implementing autonomous routines, there are a range of complementary sensors that can be used alongside the positioning sensors to further improve the complexity or reliabilty of the routines.

Limit Switches & Bumper Switches

Limit switches are small sensors that detect whether a small, flexible metal lever is being compressed.
Similarly, Bumper Switches also detect compression – but uses a button-like structure instead.

An image of a limit switch [22]

An image of a bumper switch [22]

Uses:
- Stop action once limit is reached, i.e. a lift reaching minimum extension
- Stop robot once contacting wall

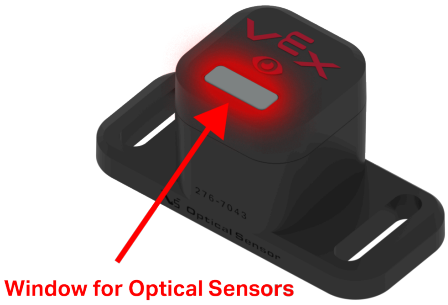
Pros	Cons
• Hardware solution *can* be simpler[5]	• Uses 3-wire ADI port • Code based solutions exist and are generally easier

[5] Although most likely not.

Optical and AI Sensors

Optical Sensors

Optical sensors are small sensors that detect 2 things: distance and colour. Distance will only tell the program roughly how far the object is (e.g. near or far) and so cannot be used as a replacement for the distance sensor; however, the colour sensing can be a very useful tool, especially in High Stakes where game objects are colour coded for alliances.



An image of the V5 Optical Sensor and it's light window [23]

A screenshot of the Optical Sensor's menu [23]. Note the distance is not classified by a value.

> **Example**
>
> **Example Optical Sensor Usage**
> One example of how an optical sensor can be used is **colour sorting**; if the robot has an intake, where rings are picked up from the floor and transported onto a mogo, an optical sensor can be used to trigger something that interupts the rings' travel – therefore not scoring it on a mogo.

> **Note**
>
> While this section is on *autonomous sensors*, optical sensors can be used (like the above) during driver control to aid the driver.

Pros

- Colour sorting mechs can take significant amount of work off driver
- Colour sorting allows for some built in tolerance during the autonomous

Cons

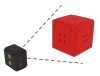
- Requires more complex programming
- Can be somewhat inconsistent, especially at longer ranges

AI Sensors

AI Sensors are the more complex, bigger brothers of the optical sensors. Similarly to an optical sensor, they can detect colours,

Image of the AI vision sensor [24]

however, they also have a larger field of view, are capable of detecting multiple contrasting objects, including their size, distance and angle – and can even be trained to detect certain objects.



An example of how AI vision sensors can detect 3D objects [24]

Pros	Cons
• Aligning with objects during autonomous • More precise than colour and distance sensors	• Much more complex programming • Some objects have to be trained, therefore much **more tuning**

Brainstorm Summary

In summary, many (oh so many) sensors can be used in autonomous, to provide the basis of routines, or to aid with peripheral functions.

Going Forward

We can now begin to narrow down the options based on what we as a team want to target, and how we can most efficiently improve our autonomous routines.

Deciding on Autonomous Sensors

With so many options to choose from, it is preferable to group some typically used (for positioning) combinations and decide from those, this is especially prominent as some sensors are useless without other sensors. Complimentery will be decided after.

Positioning Sensor Combinations

Below are the possible sensor combinations to choose from.

- Drivebase IMEs & IMU – Odometry
- IMEs only – Odometry
- GPS Sensor – VEX Global Positioning
- IMU only – INS
- 2x rotation sensor odometry pods & IMU – Odometry
- 3x rotation sensor odometry pods – Odoemtry
- 2x OSEs sensor odometry pods & IMU – Odometry

Descision Matrix

Weights

The matrix is weighted like so:

- Cost: 5 (we don't want to spend money)
- Ease of Build: 2, (complex builds are less of a problem)
- Ease of Program: 1 (complex programming is not a problem)
- Reliabilty: 3 (important but quick fixes can make it manageable)
- Accuracy: 4 (we want accurate autons)

	Cost	Ease of Build	Ease of Program	Reliabilty	Accuracy	Total
IMEs & IMU	25	10	3	9	8	55
IMEs Only	25	10	3	6	8	52
GPS Sensor	15	4	3	3	4	29
IMU Only	25	10	2	3	4	44
2x R-Odom Pods & IMU	10	4	4	15	20	53
3x R-Odom Pods	5	4	4	15	20	48
2x OSE-Odom Pods & IMU	25	4	4	15	20	68

> **Note**
>
> A perfect cost score (25) means we already have it.

Final Decision

We ultimately chose to go with 2x OSE based odometry pods with an IMU, mostly because they provide accurate results, and we already have all the parts.

Complimentery Sensors

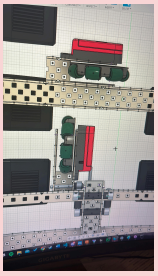
Final Decision

Ultimately, we have for now decided against using any complimentary sensors, mostly due to cost and existing (less expensive) options. We have however bought a optical sensor for later use.

Build Complete

Build Overview

Before we started to build Building the odometry pod was fairly easy, we had to cut a c channel so the inside was removed then add the (half) wheels with spacing on an LSS into an OSE.




? Manipulating Field Elements: Rings 2024/09/16

The Need For a Ring Mech

Scoring and descoring points in VEX HiS requires manipulating and placing rings onto stakes: either mogos or wall stakes. This means that in order to score points and therefore have a chance at winning we need a mechanism, which is capable of scoring rings onto mogos and eventually onto wall stakes. Our current game analysis and overall team evaluation posits that High Stake is not worth designing a mechanism for (despite its lucrative point scoring potential) due to the requirements of having a T3 climb on our bot. Overall we want a mechanism that can:

- Intake rings
- Elevate rings upwards
- **Reliably** score rings onto mogos

– Potentially score rings onto alliance and wall stakes

The requirements for our ring mech are much more rigorous than other components, which is due to its importance in allowing us to score and win matches. Currently, scoring onto wall and alliance stakes is more of a stretch goal than a necessity at the moment.

Example

Additional Motivation

Throughout MOA, every single team, which made elimination had a ring intake. This has helped us realise that competing without a ring mechanism would be a catastrophic blunder.

Brainstorming First Stage Intake for V1

2024/09/16

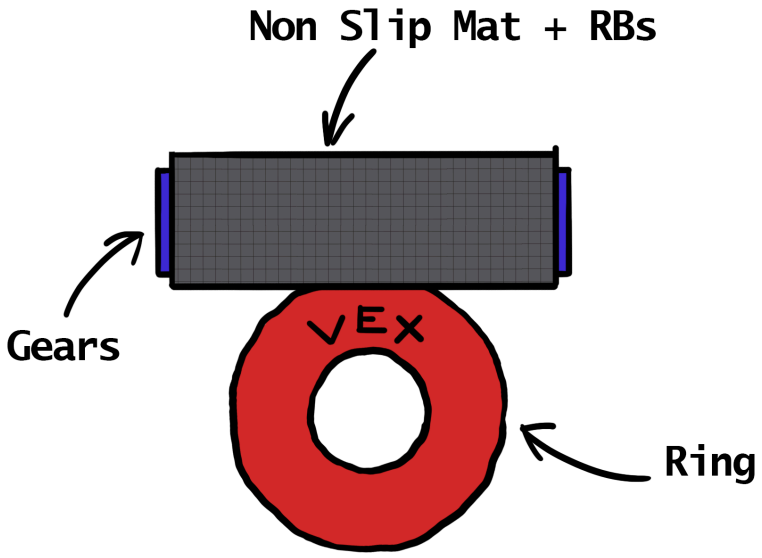
First Stage Intake

The first requirement is being able to intake rings into the bot. There are several ways this can be done using either flex wheels, claw mechanism, intake rollers or rubber bands + a non slip mat. In the context of our robot any solution that uses more than 22W is **not possible** because our robot's drivetrain uses 66W out of our budget already and we are not willing to comprimise our drive train's performance.

Rubber bands + Non-slip Mat First Stage Intake Design

Our first consideration is the use of a mechanism comprising two gears, a low/high strength shaft, rubber bands and a non slip mat. This solution is similar to a flex wheel based solution because the rubber bands provide flexibility but don't grip as much as flex wheels since they don't conform to the ring shape.
Here are a list of pros and cons for rubber bands.

Pros	Cons
• Fast intake speed • Works well with funnels to increase consistency	• Inconsistent • Rubber bands can snap



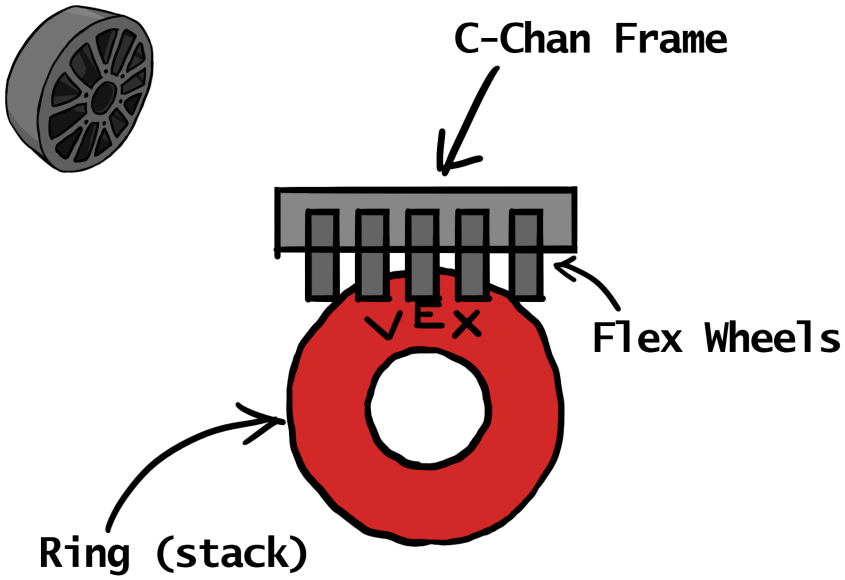
Photoshop sketch of a non-slip mat intake

Flex Wheel First Stage Intake

A Flex Wheel intake works by having several flex wheels in series on a high or low strength shaft powered by a motor, either directly powered or with a chain. It is similar to the RB/Non slip mat solution but since the wheels conform to the ring shape the total surface area

---

contacting with the ring is greatly increased. An increase in surface area means that there is an increase in the total traction, which increases the total gripping power. Overall, this means that it is easier to intake rings whilst also being fairly consistent from a variety of angles. Another advantage is that unlike with rubber bands we can control where the grip is concentrated on the rings, which is important since the rings are torus shaped meaning wheels in the centre are less useful since the places where the rings can be gripped changes are they are rings are being intook. Additionally, we can further control how much deformation the flex wheels undergo since they are various durometers of wheels, which mean that we can control the stiffness of the intake.
Below are a list of pros and cons for a flex wheel based intake:



Flex wheel intake

Pros	Cons
• Fast intaking speed • Consistent from front • Fairly consistent from various angles[1] • Good traction since wheels are compressive • Level of compression can be controlled • Positions of grip can be controlled	• Requires a dedicated motor • Risk of motor burnout if intake is used too much

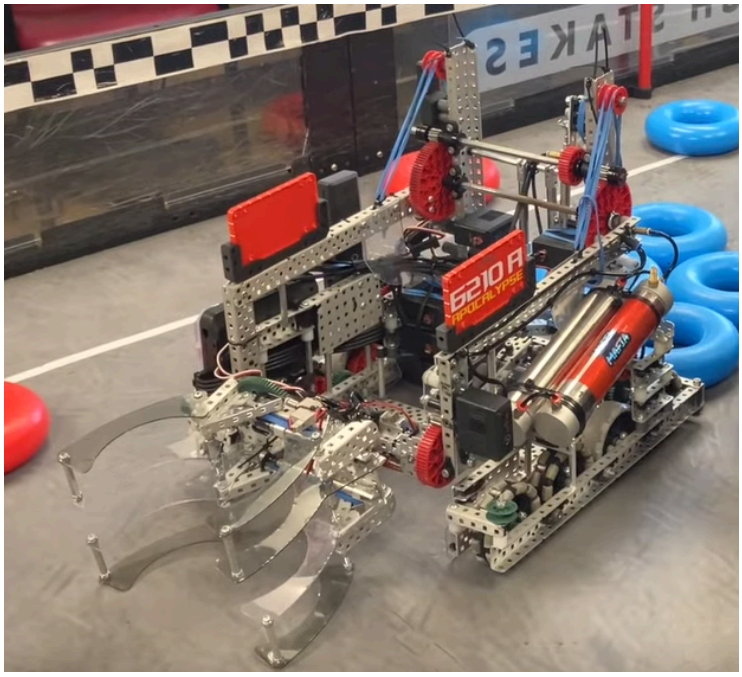
[1]Can be made more consistent from variety of angles by using polycarb funnels pointed into the intake.

Brainstorming First Stage Intake for V1

2024/09/16

Wrist/Elbow Mechanism

Another possibility for a ring intake is a claw and wrist mechanism, which utilises a claw - made using polycarb pincers and pistons - in conjunction with a series of motors on pivots to create a robotic arm mechanism, which can pick up a ring and lift it up to various heights and positions and also flip the ring stack held upside down. This solution can allow you to pick up to 2 rings in front of the robot and place them directly onto a mogo or a wall stake using various preprogrammed stakes. If tuned properly, this can be made consistent. However, this solution creates a very difficult programming task as all the motor positions and transitions between states must be mapped out and addressed. In addition, this mech requires at least 2-3 motors, which is **at least** 11W out of the power budget. Whilst the ring mechanism is **important**, this is a high cost compared to other solutions, which can do the same job for fewer motors. A good example of this solution implementation is *6310A's first bot of the season* .
Initial analysis of wrist style solution:



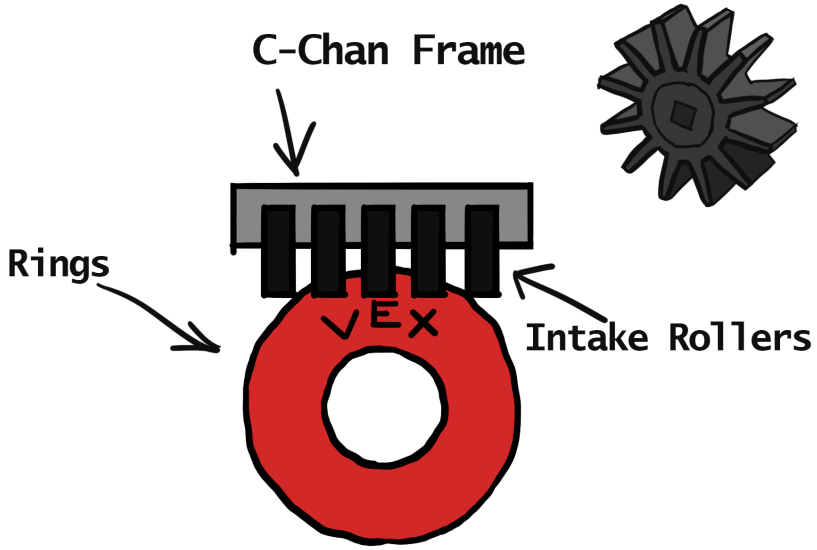
Wrist Mech Example

Pros	Cons
• Can be made consistent • Theoretically doesn't need a second stage mechanism • Fairly simple	• Requires at least 2-3 motors • Struggles to initially grab the ring • Slow compared to other solutions • Risk of motor burnout if intake is used too much • Requires complex programming due to many states

Intake Rollers

Similarly to the Flex Wheels you can use intake rollers on a high strength or low strength shaft with a direct or chain powered motor. Unlike flex wheels, intake rollers don't compress, which means they have less surface area contacting the rollers and therefore less traction. Overall this means that intake rollers have less grabbing and gripping power than a flex wheel intake. Additionally, there's only one size of an intake roller, which means we have less control over the size and vertical positioning of the intake.
Below are the benefits and drawbacks of Intake Rollers:

Pros	Cons
• Fast intake speed • Fairly consistent • Works well with funnels to increase consistency	• Intake rollers are eclipsed by flex wheels in terms of performance • Not as high traction as flex wheels • Only one size



How Intake Rollers could look

Second Stage Intake

Once we have the ability to intake rings into the robot we need to consider how we want to get rings scored onto mobile goals. If we go with a wrist mechanism, this isn't needed. However, we shall look at second stage anyway since we haven't yet decided what our first stage will be. The main two design archetypes to consider are a **hood** mechanism or a **hook** mech.

Brainstorming First Stage Intake for V1

2024/09/16

Hook mechanism

A hook mechanism works by using a motor powering a vertical or diagonal chain drive, which has hooks typically made using either custom cut polycarbs. The hook catches the inside of the ring and the chain drive acts like an escalator up towards the position of where a clamped mogo should be and directly places a ring on it. This will require some tuning in order to get the alignment of the mogo and the placement of the rings within desired tolerance (as low as possible).
Considerations with this solution:

Pros	Cons
• Fast • Consistent if tuned • Can theoretically be used as first and second stage intake as seen with *1000A at MN signature* and *This prototype by JHAWK/9 Motor Gang*	• Tuning can take time • Struggles with “cycling” if hook is misaligned[2]

Hood mechanism

Another option is using a conveyor belt in conjunction with a hood mechanism in order to outtake directly onto a mogo. Like a hook mechanism this also requires a lot of tuning in order to get right. A hood can be used in conjunction with a *bucket mechanism* in order to have rings to get on wall/alliance stakes.
Below is some initial evaluation:

Pros	Cons
• Consistent if tuned • Can double as wall stake mechanism	• Tuning can take time

[2] “However this cycling issue can be overcome with clever *design* “

Deciding on an Intake for V1 2024/09/16

Deciding on our first stage intake

Ultimately a reliable design that can **consistently** and **accurately** intake and put rings onto stakes and mogos is our top priority when choosing a design for our first stage intake. A design that requires fewer motors and is fast also holds signifigant weighting in our decision. Ultimately, for each design we are going to be comparing:

- Consistency
- Traction
- Speed
- Motor requirements
- Build complexity

	Consistency	Traction	Speed	Motor Requirements	Build Complexity	Total
Rubber Bands/ Non-slip mat	3	3	3	3	5	17
Flex Wheels	5	5	4	4	2	20
Wrist Mechanism	1	2	1	1	1	6
Intake Rollers	3	3	3	3	4	16

Deciding on our second stage intake

Since we are not going for a claw mech a second stage will be needed. For our second stage we are looking for a reliable solution but also for a solution that is fast and is preferably light. This decision is tricky since there is a lot of *discourse* and mixed signals in the VEX community about which solution is better.

	Consistency	Speed	Weight	Total
Hood Mech	4	3	3	10
Hook Mech	4	4	4	12

Final Decision

From our analysis and our preconstrained requirements (outlined in the brainstorm section), a flex wheel intake is by far the best choice for an us to implement for our first stage because it is has excellent traction and grip, its ability to perform the task is consistent, it only needs one motor and we can position the flex wheels in different configurations to control points of contact when intaking the ring. A flex wheel first stage necessitates the need for a second stage intake, in order to score rings onto the mogos. For this we decided to go ahead with a hook based solution since it allows us since it is fairly fast, can be made consistent if tuned properly and is also lighter than a hood mechanism. Whilst in brainstorm one of the benefits of a hood is its higher affinity to a bucket mech this is *still possible* with a hook based solution. However for the time being we are choosing not to dedicate time designing a wall stake mech.

Current Plans:

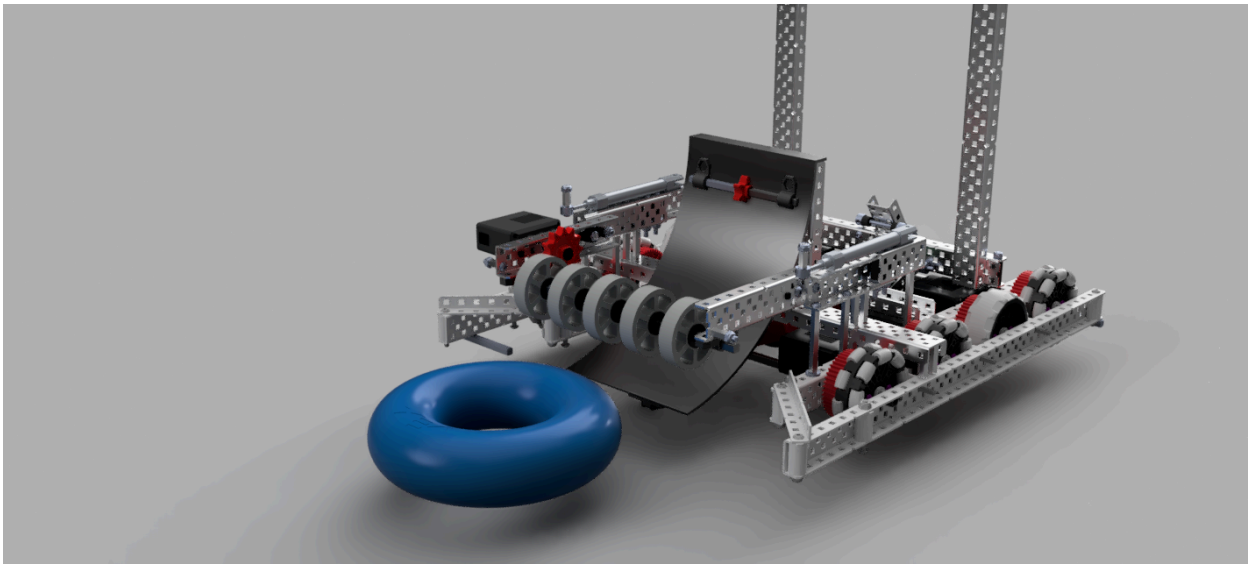
- Flex Wheel First Stage
- Hook mech
- No current plans for wall stakes

Building the Intake for V1

2024/09/21

CADing the Intake:

As with all other elements of our robot our intake started life as a digital model on the screen so taht we could think through how it will work in person, but as we identified that the ring would have to go through a 2 stage process we decided to split this CAD into two parts, first focusing on the intial rollers that bring the ring into our robot, the process for CADing this involved slightly more skill than the other elements of the robot due to the necessity that the angles of slope and flex wheel height is correct,



Here we can see the detailed CAD model of the initial stage of our intake which gives us a detailed plan of which to construct the physical mechanism off of, as can be seen this design looks much like the designs of previous seasons due to similarities from year to year in how game elements respond to manipulation.

Constructing the Intake:

As with all elements of our robot there are clear steps to be followed to build our intake and these are shown here:

Parts list:

- 1 x 2 x 15 C-Channel (x3)
- 1 x 2 x 15 L-Channel
- 0.5in Spacers (x12)
- 0.25in Spacers (x12)
- 1.5in Screws (x14)
- 2in Flex Wheels (x4)
- 8in High Strength Shaft
- Motor with blue cartridge
- High Strength Spacers (x15)
- High Strength Pillow Bearing (x2)

Designed by: Jonah Fitchew
Witnessed by: Daniel Dew

Building the Intake for V1 2024/09/21

Step 1

Construct the C-Channel frame for the intake to be built within, we used the same method as the drivetrain “squaring” to make sure each and every angle was 90 degrees and all bracing will work as intended, we also utilised boxing, which uses spacers on every screwlink on the frame to ensure stability and make sure the C-Channel does not bend.

Step 2:

We then added the bearings and more bracing to the front of the intake as a “ram bar” this is because in match situations often the robot is subjected to a large amount of force and abuse at the front, this bracing enables us to not break half way through a game. the bearings allow for lower friction which means the intake can work more desireably.

Step 3:

Finally we added the flex wheels and the motor to the rest of the frame, we used a 6T sprocket and chain to link the motor to the axle that the flex wheels are on on a 1:1 ratio, to make sure the intake is running at a suitably fast speed to pick up rings at the desired rate.

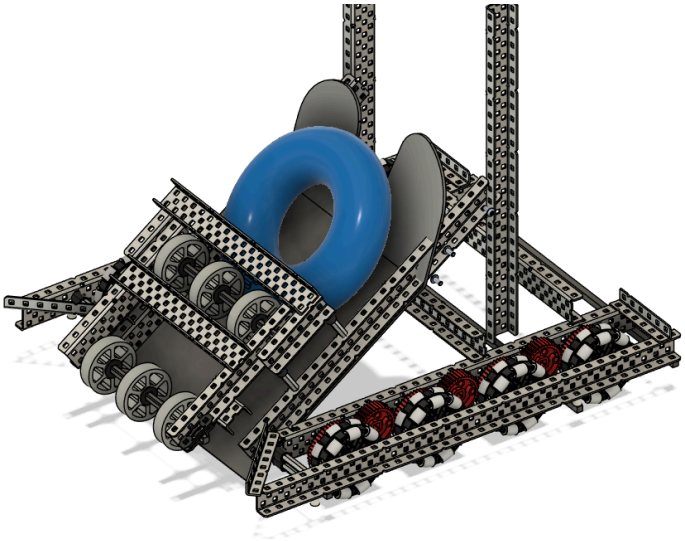
Why we built like this:

- We used a C-Channel frame because this frame is lightweight and rigid allowing for weight savings without the cost of a flimsy mechanism, we also have a lot on hand so extra purchases were not necessary
- We used high strength shafts since they are incredibly rigid and help mitigate bending which could cause friction since flex wheels already have an insert to convert their hexagonal centre to one that would fit on a high strength shaft, should we have used low strength we would have to add a further insert which could be the source for more friction
- We went with flex wheels because we had them on hand from last season which **minimises costs**. Also as rings are made of a similar material to triballs we felt that the grip flex wheels provide would yield optimal performance since they worked well in last year’s game. Additionally they are also fairly low profile and lightweight.

Rebuilding the V1 Intake 2024/09/23

Rebuild

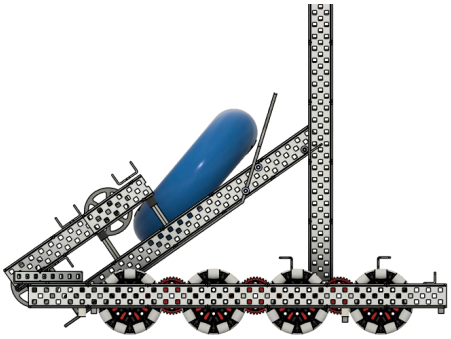
We decided to rebuild the intake due to the fact that the transition between first and second stage intake would be very difficult to fit in, due to the positioning of the first stage and the bracing. As a result I CADed a theoretically sound intake to accommodate it all without issue.

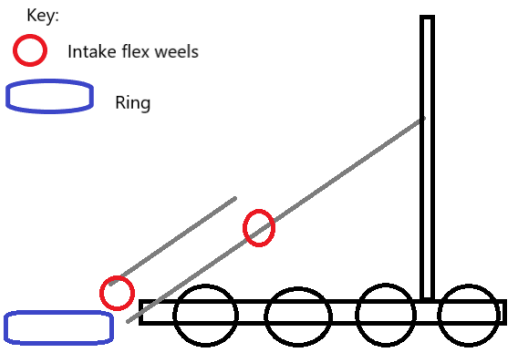


CAD

First Stage

The plan was for the first stage one rollers to bring a ring up a polycarb ramp with a depression in the end, allowing hooks to pass and pick up the ring in the center before moving the ring up and scoring it on the high stake. The first stage was also largely based of the initial first stage intake in order to salvage that work. However during the building it became obvious that this configuration would not function. The second stage rollers would not fit in their pre-assigned position and even if they did they would not push the ring far enough up the slope for it to reach the hooks. Unfortunately by this point we had very little time left before the competition to CAD something new, build, fine tune it, and do autons, so I freestyled it with the rough frame work from the CAD.



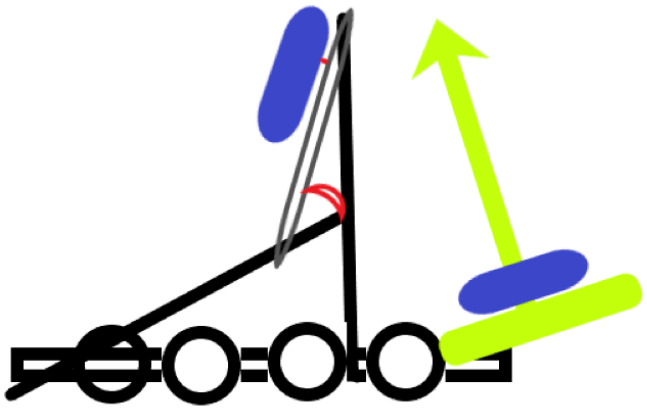


CAD *Diagram of what I want to do*

Instead of using two sets of flex weels on top we used one on the top and one on the bottom as this allowed us to use fewer flex wheels more efficiently and save on space solving our problems.

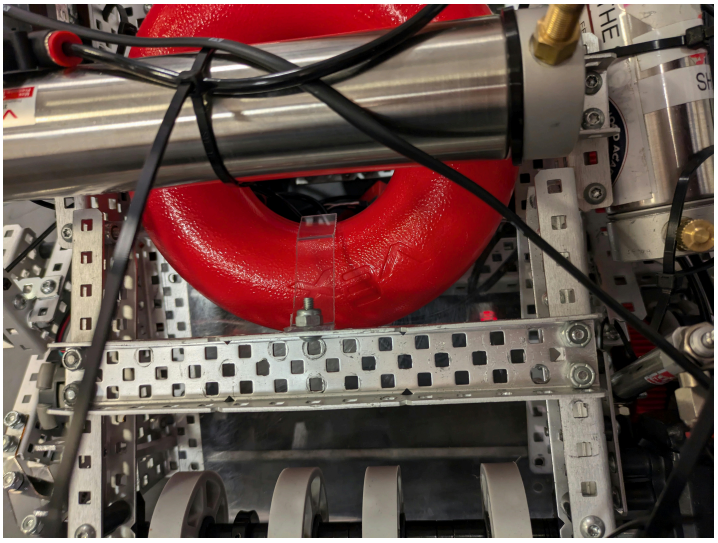
Second stage

The second stage is a hooks mech. The hooks come up from underneath hooking on to a ring, carrying it to the top and scoring it on a mobile goal. At least in theory that is how it would work.



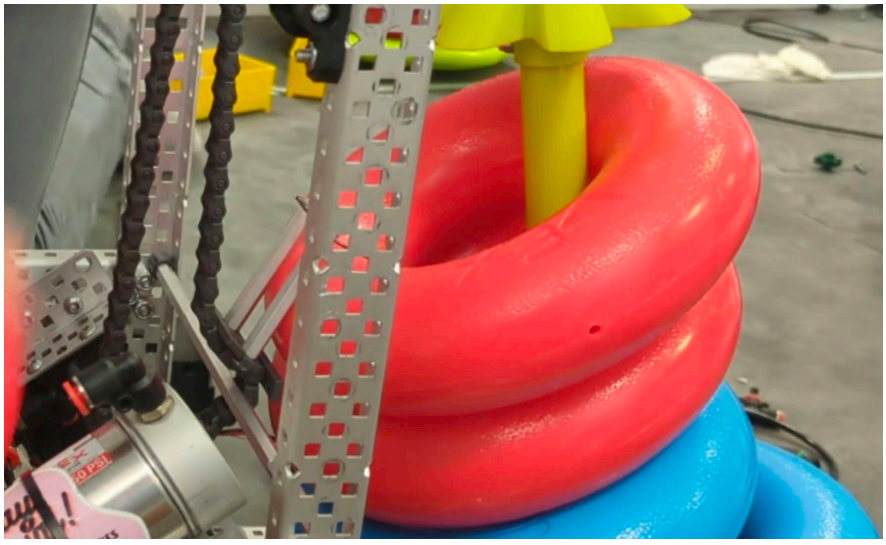
Hooks concept diagram

The practical implementation took a lot more work than creating a concept to suit our needs. Our first Issue became obvious when we tried to score rings for the first time and the ring shot out the top and to the side of the Intake instead of scoring on the mogo like we had hoped. The issue was at the end of the first stage rollers, they sometimes put the ring to a point where when the hooks would come along they would hook the bottom of the ring instead of the center of it and became more like a scoop than a hook. to solve this first issue I built a flexible hard stop which held the rings in the perfect position to be hooked through and the hooks had enough force to move the ring from where it was hard stopped. The hard stop was a clever thin piece of polycarb on the first stage intake that would hook through the center of a ring going up the intake and hold it there until enough pulling force is applied by the hooks.



Flexible hard stop

The second main issue we encountered when experimenting with our old intake was when more than 3 ring were scored on the mogo the hooks would get stuck and would not be able to complete a full rotation. This issue would be truly catastrophic unresolved as it would limit your possible points scored to 5/8 or require us to make the intake go backwards and forwards instead of cycling properly.



Demonstration of hooks catching on rings

Initially I was stumped by this problem. When I turned to see how other teams solved this problem as I knew more than a few must have encountered it, I found very little useful information so I defined the problem, it was hooks becoming stuck on the rings, and the reason for the problem: the rings are too close to the path of the hooks. The obvious solution is to push them away but how we would achieve this was the problem. In the end I used two

sheets of poly carb bent at a right angle to the intake with a slope on the top allowing the rings to slid off smoothly. It worked.

Testing the V1 Intake 2024/09/24

Introduction

In order to confirm we have met our success criteria(as defined by the identify page) we ran tests to confirm consistency and efficiency.

Tests

We ran 3 sets of tests. Making note of how often the ring was successfully scored in comparison to the number of rings it intook.

- For the first test, we fed 12 rings into the intake to try and gauge consistency.
- For the second test we placed 12 rings in a line on the field and drove the robot forward to see if that had an effect on the consistency.
- We layed out the field in a game configuration and drove the robot in the alloted match time.

Results:

Test no.	% success rate			
	Reading 1	Reading 2	Reading 3	Average
1	58.3	66.7	66.7	62.2
2	75	66.7	75	72.2
3	73.3	72.5	73.8	72.2

Analysis:

As we aimed for as close to 100% accuracy as possible, these results that we recieved were unacceptable, failing to score between 30-40% would significantly negatively impact our game performance. Interestingly the robot scored nomonaly(10%) better when in motion compared to at rest. Understanding why this is the case can lead us to more consistant results.

Close slow motion video analisis

After recording and watching a few slow motion videos we discovered a pattern in the movement of the rings when accelerating vs when stationary and came up with a hypothesis based on the pattern.

Testing the V1 Intake

2024/09/24

Hypothesis

When accelerating, the ring is rotated in such a way that they are scored fully across the mobiile goal at a lower gradient whereas when the rings are not rotated at such a high gradient, they score a lot less consistently and often don't completely encircle the stake and get stuck at a vertical angle at the top. This is due to the rings having a more constant center of rotation, allowing them to complete a fuller ark.

Solutions

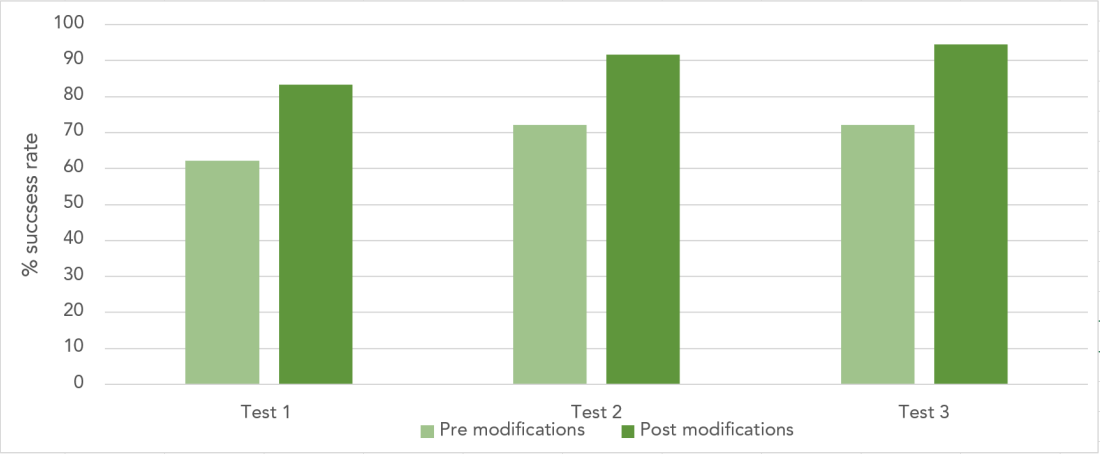
We thought of two possible solutions in order to broaden the ark the ring traveled.

- The first solution is to add a tensioner to the intake chain causing it to slope before going back to increase the length of the ark.
- The other solution we came up with was to curve the hook giving the rings a larger center of rotation and therfore a smoother, larger ark.

Second set of tests validating solutions

Test no.	% success rate			
	Reading 1	Reading 2	Reading 3	Average
1	83.3	91.7	75	83.3
2	91.7	83.3	100	91.7
3	94.6	93.2	96.0	94.6

Graph of comparitive ring scoring consistency



Designed by: Aubert
Witnessed by: Daniel Dew

Conclusion

As is demonstrated in the graph above, implementing our solutions and widening the ark significaly improved scoring consistency raising it by around 20%, which is much closer to the goal of 100% and well into acceptable ranges.

Programming Objecives

As covered in our 'Programming Approach' design process, there are certain objectives that a programmer must consider when creating a robot's program. These include:

- Driver/Operator Control
  - Somewhat self explanatory, the inputs from the controller must influence how the robot actuates its systems during the 1:45min drier control period
- Autonomous
  - Autonomous routine(s) must be programmed into the robot to be used during autonomous period
- Ease of setup
  - Before a match, the correct autonomous routines and possibly adjustable values must be set up – the programmer must consider how this is done (e.g. GUI, through code changes etc.)
- Ease of testing
  - During testing (not in competition) the robot must be bleto be easily tested – this could simply mean it is easy to change the key values in code, or could be more complex in the form of a GUI or CLI.
- Maintainability
  - This has been discussed in detail in preious pages, essentially, the programmer must keep ease of use and maintainabilty to a high standard when programming.

Orginisation is key.

> **Quote**
>
> Code is read much more often than it is written.
>
> — Guido Van Rossum, Creator of Python

The Blank Project

After creating a blank project using PROS, it supplies some defualt functions that can be used to create the program.

```cpp
#include "main.h" // -> includes the PROS API, to be used throughout the program.
/**
 * Runs initialization code. This occurs as soon as the program is started.
 *
 * All other competition modes are blocked by initialize; it is recommended
 * to keep execution time for this mode under a few seconds.
*/
void initialize() {}
```

```
/**
 * Runs while the robot is in the disabled state of Field Management System or
 * the VEX Competition Switch, following either autonomous or opcontrol. When
 * the robot is enabled, this task will exit.
 */
void disabled() {}
/**
 * Runs after initialize(), and before autonomous when connected to the Field
 * Management System or the VEX Competition Switch. This is intended for
 * competition-specific initialization routines, such as an autonomous selector
 * on the LCD.
 *
 * This task will exit when the robot is enabled and autonomous or opcontrol
 * starts.
 */
void competition_initialize() {}
/**
 * Runs the user autonomous code. This function will be started in its own task
 * with the default priority and stack size whenever the robot is enabled via
 * the Field Management System or the VEX Competition Switch in the autonomous
 * mode. Alternatively, this function may be called in initialize or opcontrol
 * for non-competition testing purposes.
 *
 * If the robot is disabled or communications is lost, the autonomous task
 * will be stopped. Re-enabling the robot will restart the task, not re-start it
 * from where it left off.
 */
void autonomous() {}
/**
 * Runs the operator control code. This function will be started in its own task
 * with the default priority and stack size whenever the robot is enabled via
 * the Field Management System or the VEX Competition Switch in the operator
 * control mode.
 *
 * If no competition control is connected, this function will run immediately
 * following initialize().
 *
 * If the robot is disabled or communications is lost, the
 * operator control task will be stopped. Re-enabling the robot will restart the
 * task, not resume it from where it left off.
 */
void opcontrol() {}
```

Adding Packages

One of the huge advantages of PROS is the abilty to add community developed packages into the code. We as a team will add 2 packages to significantly improve our program.

> **Warning**
>
> **DISCLAIMER**
>
> We as a team *cannot* claim responsibilty for the development of these packages. They are both developed by individual open source communities.
> However, before we decided to use these packages, we all researched and gained understanding of the core concepts that these packagas utilize, this not only drastically improves our capabilty to use them, but also ensures our adherance to the *Student-Centered Policy* .

The Packages:

- LemLib
  - *LemLib* is a library developed by a large open source community – it allows users to employ odometry alogorithims, as well as movement functions; including PID controllers and Boomerang Controllers.
  - We use a modified fork[1] of lemlib (developed by us), allowing us to implement *gain scheduling* to be used later.
- Robodash
  - Robodash implements a user friendly libvgl[2] interface, that allows us to quickly implement consoles and auton selectors.

Adding The Packages in The Project

To add the packages we can use the newly implemented remote depot feature from the PROS CLI.

For LemLib:

```zsh
pros c add-depot LemLib https://raw.githubusercontent.com/LemLib/LemLib/depot/stable.json # adds LemLib stable depot
pros c apply LemLib # applies latest stable version of LemLib
```

For Robodash:

```zsh
pros c add-depot robodash https://raw.githubusercontent.com/unwieldycat/robodash/depot/stable.json
```

[1] A branch, in this case not public
[2] The GUI library that PROS implements for the brain's screen

File structure

Our program will be split up into different sections for the sake of orginisation:

```
|
|-- [vscode files/folders]
|-- include
|   |-- [defualt pros header folders]
|   |-- lemlib -> lemlib header files
|   |-- robodash -> robodash header files
|   |-- usr -> created by us
|   |   |-- autons.h -> header files for all auton routines
|   |   |-- robot.h -> contains 'robot' class
|   |   |-- utils.h -> header file for utility functions
|-- src
|   |-- robotConfig
|   |   |-- globals.cpp -> where all global values are initialized
|   |   |-- intis.cpp -> initializer functions for robot
|   |   |-- tuning.cpp -> where tuning loops and functions are contained
|   |-- autons.cpp -> where auton routines are defined
|   |-- main.cpp -> all competition functions (opcontrol(), autonomous(), etc.)
|   |-- utils.cpp -> where util functions are declared
```

Class Based Program (robot.h)

To improve maintainabilty and orginisation, we will be implementing a class based program; this involves us using a header file that contains all functions (contained in subclasses) to operate the robot and its subystems. These functions can then be called and used in any other file.

robot.h:

```h
// includes ---
#include "api.h"
#include "lemlib/api.hpp"
#include "lemlib/chassis/chassis.hpp"
#include "lemlib/chassis/trackingWheel.hpp"
#include "pros/adi.hpp"
#include "pros/imu.hpp"
#include "pros/motor_group.hpp"
#include "pros/motors.hpp"
#include "robodash/api.h"
#include "robodash/core.h"
#include "robodash/views/console.hpp"
#include "robodash/views/selector.hpp"
// ---


namespace Types {
    // specifies how intake is actuated
```

```
    enum IntakeActionType {
        BOTH,
        FIRST,
        SECOND
    };
}
using namespace Types;
class Robot {
    public:
        class Auton { // contains autons
            public:
                class Tuning {  // contains functions to tune autonomous (PID) values
                    public:
                        static rd::Console AutonTuning;
                        static const void TuningLogicLoop();
                        static lemlib::ControllerSettings latController;
                        static lemlib::ControllerSettings angularController;

                        //TODO (non-urgent)
                        static const void DriveCurveTuning();
                        static lemlib::ExpoDriveCurve driveCurveLateral;
                        static lemlib::ExpoDriveCurve driveCurveAngular;
                        //--

                        static rd::Selector Tuningselector; // selector for test autons
                };
            static rd::Selector autonSelectorMain; // selector for game autons
        };
        class Motors { // contains all motor-based variables
            public:
                static const lemlib::Drivetrain drivetrain;
                static pros::Motor Intake1st;
                static pros::Motor Intake2nd;
                static pros::MotorGroup fullIntake;
                static bool DTDirection; // direction of 'forward' for drivetrain (not implemented)
            protected:
                static pros::MotorGroup leftMotors;
                static pros::MotorGroup rightMotors;
        };
        class Sensors { // class for sensors
            public:
                static lemlib::OdomSensors sensors;
                static pros::IMU imu;
            protected:
                static lemlib::TrackingWheel verticalTracking;
                static lemlib::TrackingWheel horizontalTracking;
        };
        class Pneumatics { // class for pneumatics
            public:
                static pros::adi::Pneumatics mogoMech;
                static pros::adi::Pneumatics intakeLifter;
                static pros::adi::Pneumatics doinker;
```

```
    };
    class Actions { // contains macros/actions for robot
        public:
            static void setMogoFor (bool extended, float time, bool async=true);

            static void setIntake(int direction, IntakeActionType stage);
            static void runIntakeFor(int direction, IntakeActionType stage, float time, bool async = true);


            static void setIntakeLifterFor(bool extended, float time, bool async=true);

            static void jiggle(float time); // jiggles robot (may be useful?)

    };
    class Init { // intializer functions
        public:
            static void initAll();
        protected:
            static void initPIDs();
            static void initDriveCurves();
    };
    class Logging {/* WIP */};
    static pros::Controller master; // controller
    static lemlib::Chassis chassis; // main chassis
};
```

Declaring global values (globals.cpp)

To avoid null pointers, global variables must be decalred in one file:

cpp

```cpp
#include "main.h"
#include "lemlib/api.hpp"
#include "pros/abstract_motor.hpp"
#include "pros/adi.hpp"
#include "pros/motor_group.hpp"
#include "usr/robot.h"
#include "robodash/api.h"
#include "usr/autons.h"
using namespace pros;
using namespace lemlib;

pros::Controller Robot::master (pros::E_CONTROLLER_MASTER);
rd::Console Robot::Auton::Tuning::AutonTuning ("PID Tuner");

lemlib::ControllerSettings Robot::Auton::Tuning::latController (10, // proportional gain (kP)
                                0, // integral gain (kI)
                                3, // derivative gain (kD)
                                3, // anti windup
                                1, // small error range, in inches
                                100, // small error range timeout, in milliseconds
                                3, // large error range, in inches
```

```
                                500, // large error range timeout, in milliseconds
                                20 // maximum acceleration (slew)
);
lemlib::ControllerSettings Robot::Auton::Tuning::angularController(2, // proportional gain (kP)
                                0, // integral gain (kI)
                                10, // derivative gain (kD)
                                3, // anti windup
                                1, // small error range, in degrees
                                100, // small error range timeout, in milliseconds
                                3, // large error range, in degrees
                                500, // large error range timeout, in milliseconds
                                0 // maximum acceleration (slew)
);
rd::Selector Robot::Auton::Tuning::Tuningselector (
    {
        {"Move Forwards 24''", Autons::Testers::forward24},
        {"Turn 90deg right", Autons::Testers::turn90},
        {"Boomerang 24 24 90", Autons::Testers::boomerang_24_24_90},
        {"Swing 90deg right", Autons::Testers::swing90},
        {"Cricle (exp.)", Autons::Testers::circle}
    }
);

Motor Robot::Motors::Intake1st (-11, v5::MotorGears::blue, v5::MotorUnits::rotations);
Motor Robot::Motors::Intake2nd (9, v5::MotorGears::blue, v5::MotorUnits::rotations);
MotorGroup Robot::Motors::fullIntake ({-11, 9}, v5::MotorGears::blue, v5::MotorEncoderUnits::rotations);

bool Robot::Motors::DTDirection = true; // Defualt direction is forwards

//* Left motors on ports 10, 9, 8; Rights on 1, 2, 3; Using blue cartridges
MotorGroup Robot::Motors::leftMotors (
    {-1, -2, -10},
    MotorGears::blue,
    MotorUnits::degrees
);
MotorGroup Robot::Motors::rightMotors (
    {14, 20, 18},
    MotorGears::blue,
    MotorUnits::degrees
);
ExpoDriveCurve Robot::Auton::Tuning::driveCurveLateral (4, 6, 1.004);
ExpoDriveCurve Robot::Auton::Tuning::driveCurveAngular (4, 6, 1.016);
//* Drivetrain with track width 13.1'', Geard (down) for 450rpm, using horizontal drift of 8 due to traction wheel
usage
//TODO Measure dimensions
const Drivetrain Robot::Motors::drivetrain(
    &leftMotors,
    &rightMotors,
    13.1,
    Omniwheel::NEW_275,
    450,
    6
```

```
);

adi::Pneumatics Robot::Pneumatics::mogoMech {'E', false};
adi::Pneumatics Robot::Pneumatics::intakeLifter {'G', true}; // not built
adi::Pneumatics Robot::Pneumatics::doinker {'h', false}; // not built

Imu Robot::Sensors::imu (21); // IMU on port ?

adi::Encoder vertEncoder('A', 'B');
adi::Encoder horiEncoder('C', 'D');

//TODO measure distances
TrackingWheel Robot::Sensors::verticalTracking (&vertEncoder, Omniwheel::NEW_275_HALF, 0);
TrackingWheel Robot::Sensors::horizontalTracking (&horiEncoder, Omniwheel::NEW_275_HALF, 0);

OdomSensors Robot::Sensors::sensors (&verticalTracking, nullptr, &horizontalTracking, nullptr, &imu);

lemlib::Chassis Robot::chassis (
    Motors::drivetrain,
    Auton::Tuning::latController,
    Auton::Tuning::angularController,
    Sensors::sensors,
    &Auton::Tuning::driveCurveLateral,
    &Auton::Tuning::driveCurveAngular
);
```

Utils

Utils are useful functions that can be used throughout the program to manage certain aspects.

utils.h:

```h
#include <string>
namespace utils { // namespaces for utils
    void save_value(std::string name, float value); // saves float to file on sd card
    float load_value (std::string name); // loads float from file on sd card
}
```

utils.cpp (declaring):

```cpp
#include "usr/utils.h"
#include "fmt/core.h"
#include "main.h"
#include <cstdio>
#include <cstring>

void utils::save_value(std::string name, float value) {

    std::string fullPath = fmt::format("/usd/{}.txt", name);
```

```
    FILE* value_file = fopen(fullPath.c_str(), "w");
    fputs(std::to_string(value).c_str(), value_file);

    fclose(value_file);
}

float utils::load_value(std::string name) {

    std::string fullPath = fmt::format("/usd/{}.txt", name);

    FILE* value_file = fopen(fullPath.c_str(), "r");

    if (value_file == nullptr) {
        return 0;
    }
    char buf[50];
    fread(buf, 1, 50, value_file);
    fclose(value_file);

    return std::stof(buf);
}
```

Autons

> **Note**
>
> As of right now, autons are WIP – only test autons are implemented.

autons.h:

```h
namespace Autons {
    class Testers {
        public:
            static void forward24(); // moves robot forward 24 inches
            static void turn90(); // turns 90deg right
            static void swing90(); // 'swings' 90deg right
            static void boomerang_24_24_90(); // uses boomerang controller to go to (24, 24) inches at heading 90deg
            static void circle(); // (in theory) moves bot in full circle.
    };
    // WIP
    void supportTouchLadder();
    void support();
    void rush();
    // TODO: more?
}
```

autons.cpp

```cpp
#include "lemlib/chassis/chassis.hpp"
#include "usr/robot.h"
#include "api.h"
#include "lemlib/api.hpp"
#include "usr/autons.h"
using namespace pros;

//* For Tuning
void calibrate () {
    Robot::chassis.calibrate();
    pros::delay(1000);
    Robot::chassis.setPose({0,0,0});
}
void Autons::Testers::forward24() {
    calibrate();
    Robot::chassis.moveToPoint(0, 24, 1000);
    Robot::chassis.waitUntilDone();
}
void Autons::Testers::turn90() {
    calibrate();
    Robot::chassis.turnToHeading(90, 1000);
    Robot::chassis.waitUntilDone();
}
void Autons::Testers::boomerang_24_24_90 () {
    calibrate();
    Robot::chassis.moveToPose(24, 24, 90, 1250, {.lead=0.81});
    Robot::chassis.waitUntilDone();
}
void Autons::Testers::swing90() {
    calibrate();
    Robot::chassis.swingToHeading(90, lemlib::DriveSide::RIGHT, 500);
    Robot::chassis.waitUntilDone();
}
void Autons::Testers::circle() {
    calibrate();
    Robot::chassis.moveToPose(0, 24, 270, 4000, {.lead=0.6});
    Robot::master.rumble(".");
    Robot::chassis.moveToPose(-24, 0, 180, 1000, {.lead=0.81});
    Robot::chassis.moveToPose(0, -24, 90, 1000, {.lead=0.81});
    Robot::chassis.moveToPose(24, 24, 0, 1000, {.lead=0.81});
    Robot::chassis.moveToPose(0, 24, 90, 1000, {.lead=0.81});
    Robot::chassis.waitUntilDone();
}
lemlib::Chassis* chassis = &Robot::chassis;
//* Game autons
void Autons::support() {
    chassis->setPose({-50, 41, 305});
    chassis->moveToPoint(-32, 28, 1000, {.forwards=false, .maxSpeed=110, .minSpeed=40});
    chassis->waitUntilDone();
    Robot::Pneumatics::mogoMech.set_value(true);
    delay(40);
```

```
    chassis->turnToPoint(-24, 48, 500);
    Robot::Actions::setIntake(1, Types::BOTH);
    chassis->moveToPoint(-26, 42, 1000, {.maxSpeed=80});
    chassis->waitUntilDone();
    delay(100);
    chassis->turnToPoint(-3, 50.5, 400);
    chassis->moveToPose(-3, 52, 90, 1000, {.lead=0.9, .maxSpeed=80});
    chassis->waitUntilDone();
    delay(100);
    chassis->moveToPose(-12, 55, 135, 800, {.forwards=false, .lead=0.5, .maxSpeed=70});
    chassis->moveToPoint(-7, 48, 1000, {.maxSpeed=90});
}
```

Robot Actions

robotActions.cpp:

cpp

```cpp
#include "lemlib/pose.hpp"
#include "main.h"
#include "lemlib/api.hpp"
#include "pros/rtos.hpp"
#include "usr/robot.h"

using namespace pros;
using namespace lemlib;

void Robot::Actions::setMogoFor(bool extended, float time, bool async) {
    if (async) {
        Task task ([=] {setMogoFor(extended, time, false);});
        delay(10);
        return;
    }
    Pneumatics::mogoMech.set_value(extended);
    delay(time);
    Pneumatics::mogoMech.toggle();
}
void Robot::Actions::setIntake(int direction, IntakeActionType stage) {
    int volts = 12000 * direction;
    if (stage==IntakeActionType::FIRST || stage == IntakeActionType::BOTH) {
        Motors::Intake1st.move_voltage(volts);
    }
    if (stage==IntakeActionType::SECOND || stage == IntakeActionType::BOTH) {
        Motors::Intake2nd.move_voltage(volts);
    }
```

```
}
void Robot::Actions::runIntakeFor(int direction, IntakeActionType stage, float time, bool async) {
    if (async) {
        Task t ([=] {runIntakeFor(direction, stage, time, false);});
        delay(10);
        return;
    }
    setIntake(direction, stage);
    delay(time);
    setIntake(0, stage);
}
void Robot::Actions::setIntakeLifterFor(bool extended, float time, bool async){
    if (async) {
        Task t ([=] {setIntakeLifterFor(extended, time, false);});
        delay(10);
        return;
    }
    Pneumatics::intakeLifter.set_value(extended);
    delay(time);
    Pneumatics::intakeLifter.toggle();
}
void Robot::Actions::jiggle(float time) {
    float start = millis();
    while (millis()-start < time) {
        Pose jigglePoint1 = Pose {chassis.getPose().x-(float)cos(chassis.getPose(true).theta)*2, chassis.getPose().y-(float)sin(chassis.getPose(true).theta)*2};
        Pose jigglePoint2 = Pose {chassis.getPose().x+(float)cos(chassis.getPose(true).theta)*2, chassis.getPose().y+(float)sin(chassis.getPose(true).theta)*2};
        Pose start = chassis.getPose();
        Robot::chassis.moveToPoint(jigglePoint1.x, jigglePoint1.y, 50, {.minSpeed=100});
        Robot::chassis.moveToPoint(jigglePoint2.x, jigglePoint2.y, 50, {.minSpeed=100});
        Robot::chassis.moveToPoint(start.x, start.y, 50, {.minSpeed=100});
        Robot::chassis.waitUntilDone();
        delay(200);
    }
}
```

Tuning

tuning.cpp

```cpp
#include "lemlib/chassis/chassis.hpp"
#include "pros/imu.hpp"
#include "pros/misc.h"
#include "pros/misc.hpp"
#include "pros/motor_group.hpp"
#include "robodash/views/console.hpp"
#include "robodash/views/selector.hpp"
#include "usr/robot.h"
#include <cstdio>
#include <cstring>
```

```cpp
#include <string>
#include "usr/utils.h"
using namespace utils;


const void Robot::Auton::Tuning::TuningLogicLoop() {
    AutonTuning.focus();
    std::cout << "here";
    float AngularP = angularController.kP;
    float AngularI = angularController.kI;
    float AngularD = angularController.kD;
    float LatP = latController.kP;
    float LatI = latController.kI;
    float LatD = latController.kD;
    float step = 0.1;
    int index = 0;
    int minIndex = 0;
    int maxIndex = 6;
    while (!master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_R2)) {
        pros::delay(20);
        AutonTuning.clear();

        latController.kP = LatP;
        latController.kI = LatI;
        latController.kD = LatD;

        angularController.kP = AngularP;
        angularController.kI = AngularI;
        angularController.kD = AngularD;

        chassis.changeAngularP(AngularP);
        chassis.changeAngularI(AngularI);
        chassis.changeAngularD(AngularD);

        chassis.changeLatP(LatP);
        chassis.changeLatI(LatI);
        chassis.changeLatD(LatD);

        AutonTuning.println("AUTON TUNING");
        AutonTuning.println("Press B to run Auton, R2 to quit, Arrows to navigate/change values, X to save values");

        AutonTuning.printf("Angular P: %f %s\n", AngularP, index==0 ? "<<" : "");
        AutonTuning.printf("Angular I: %f %s\n", AngularI, index==1 ? "<<" : "");
        AutonTuning.printf("Angular D: %f %s\n", AngularD, index==2 ? "<<" : "");
        AutonTuning.printf("Lateral P: %f %s\n", LatP, index==3 ? "<<" : "");
        AutonTuning.printf("Lateral I: %f %s\n", LatI, index==4 ? "<<" : "");
        AutonTuning.printf("Lateral D: %f %s\n", LatD, index==5 ? "<<" : "");
        AutonTuning.printf("Change step: %f %s\n", step, index==6 ? "<<" : "");

        if (master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_DOWN)) index +=1;
        if (master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_UP)) index -= 1;
        if (index > maxIndex) index = minIndex;
```

```
64      if (index < minIndex) index = maxIndex;
65
66      if (master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_LEFT)) {
67        switch (index) {
68          case 0:
69            AngularP = ((int)(AngularP*100))-((int)(step*100))/100;
70            break;
71          case 1:
72            AngularI = ((int)(AngularI*100))-((int)(step*100))/100;
73            break;
74          case 2:
75            AngularD = ((int)(AngularD*100))-((int)(step*100))/100;
76            break;
77          case 3:
78            LatP = ((int)(LatP*100))-((int)(step*100))/100;
79            break;
80          case 4:
81            LatI = ((int)(LatI*100))-((int)(step*100))/100;
82            break;
83          case 5:
84            LatD = ((int)(LatD*100))-((int)(step*100))/100;
85            break;
86          case 6:
87            step = (((int)(step*100))-1)/100;
88            break;
89        }
90      } else if (master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_RIGHT)) {
91        switch (index) {
92          case 0:
93            AngularP = ((int)(AngularP*100))+((int)(step*100))/100;
94            break;
95          case 1:
96            AngularI = ((int)(AngularI*100))+((int)(step*100))/100;
97            break;
98          case 2:
99            AngularD = ((int)(AngularD*100))+((int)(step*100))/100;
100           break;
101         case 3:
102           LatP = ((int)(LatP*100))+((int)(step*100))/100;
103           break;
104         case 4:
105           LatI = ((int)(LatI*100))+((int)(step*100))/100;
106           break;
107         case 5:
108           LatD = ((int)(LatD*100))+((int)(step*100))/100;
109           break;
110         case 6:
111           step = (((int)(step*100))+1)/100;
112           break;
113       }
114     }
115     chassis.changeAngularP(AngularP);
116     chassis.changeAngularI(AngularI);
```

```
        chassis.changeAngularD(AngularD);
        chassis.changeLatP(LatP);
        chassis.changeLatI(LatI);
        chassis.changeLatD(LatD);

        if (master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_B)){
            master.clear_line(1);
            AutonTuning.printf("Running auton...");
            master.print(1, 1, "Running auton");
            Tuningselector.run_auton();
        }else {
            AutonTuning.printf("\n");
            master.clear_line(1);
            master.print(1, 1, "Done");
        }

        if (master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_X)) {
            AutonTuning.println("Saving");
            save_value("angP", AngularP);
            save_value("angI", AngularI);
            save_value("angD", AngularD);
            save_value("latP", LatP);
            save_value("latI", LatI);
            save_value("latD", LatD);
        }
    }
    Tuningselector.focus();
}
```

Implementing the Primary Code

With all the complimentary functions built into the robot class, I can now imlement the main functions in the main.cpp file – this will include calling the selected autonomous routine and collecting and using user inputs during operator control.

Initialization

We had the foresight to declare initialize functions globally in *inits.cpp*, therefore we can just call '*initAll()*' from the primary code:

```cpp
void initialize() {
    Robot::Inits::initAll(); // runs initialization function
}
```

Operator Control

> **Note**
>
> Most functionality within opcontrol is contained in a while loop, this allows the program to constantly update values and assess states.

Moving the drive

> **Final Decision**
>
> For driver control, I (driver and programmer) have decided to use a control scheme called **single stick curvature drive**, it is very similar to single stick arcade, but the turning inputs are scaled differently depending on throttle input (essentially creating an arc with the turning) – I chose this because arcade is typically limited in terms of precise control, and tank, while it technically is easier to provide ultra precise turn circles, is not very intuitive and would be very difficult to learn in such a short timeframe. (We will evaluate this decision as the season progresses.)

Fortunately, LemLib condenses the somewhat long mathematical function into 1 function call:

```cpp
void opcontrol() {
    while (true){
        pros::delay(20) // 20ms delay so brain doesn't crash
        int leftX = Robot::master.get_analog(pros::E_CONTROLLER_ANALOG_LEFT_X);
        int leftY = Robot::master.get_analog(pros::E_CONTROLLER_ANALOG_LEFT_Y);
        Robot::chassis.curvature(leftY, leftX); // lemlib's chassis contains curvature function
        // ...
    }
}
```

Actuating Subsystems

To actuate the other subystems, if statements can be used to access the state of buttons on the controller:

```cpp
void opcontrol() {
    while (true){
        // ...
        if (Robot::master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_B)) {
            Robot::Pneumatics::mogoMech.toggle(); // toggles mogo on B press
        }
        if (Robot::master.get_digital(pros::E_CONTROLLER_DIGITAL_L2)) {
            Robot::Actions::setIntake(-1, Types::BOTH); // sets intake backwards while L2 is pressed
        } else if (Robot::master.get_digital(pros::E_CONTROLLER_DIGITAL_R2)) {
            Robot::Actions::setIntake(1, Types::BOTH); // sets intake forwards while R2 is pressed
        } else Robot::Actions::setIntake(0, Types::BOTH); // stops intake
    }
}
```

Extra features

With plenty of buttons spare, extra features can be added – for exapmple, we plan on having an arm/sweeper that can be added with another button press.
We can also use button presses to actuate autonomous for testing (we should of course avoid during competition). Instead of a single button press, however, it should be a sequence, so we avoid running it during driver practice.

Example

An example button sequence would be:
Down arrow held down AND up arrow new press

Adding to code:

```cpp
void opcontrol() {
  while (true) {
    // ...
    if (
      !pros::competition::is_connected() && // if field control (at a competition) is NOT connected
      Robot::master.get_digital(pros::E_CONTROLLER_DIGITAL_DOWN) && // if down arrow is held down
      Robot::master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_UP) // if up arrow is newly pressed
    ) autonomous() // starts autonomous
  }
}
```

Starting Autonomous

Robodash fortunately handles picking autonomous via a GUI, so the autonomous function must just run the seleced auton like so:

```cpp
void autonomous() {
  Robot::Auton::autonSelectorMain.run_auton(); // runs selected auton
}
```

Note On Testing & Iteration

Normally after implementing a large feature, such as this code – we would implement a testing campaign to verify the results of the code. However, we managed to very quickly verify that the code worked by simply running it and verifying all features worked.
Throughout the season, we will continue to update the code with autons, subsystems and additional features as we see fit.

Quote

Every great developer you know got there by solving problems that they were unqualified to solve until they actually did it.

— Patrick Mckenzie, Sofware Enginner

Goals for Autonomous

While we have considered in depth the sensors we want to use for autonomous – knowing where the robot is is only half the battle. Planning and implementing autonomous can be an extremely difficult task, even with perfect sensor usage.
The first step to a effective auton is establishing clear and understandable goals, these can be split into 2 main parts: securing ABP[1], and securing AWPs[2].

ABP

Overview

To secure the ABP, an alliance must score more points during the 15s autonomous period than the other team, while not breaking *any* rules [1]. This is a particularly difficult challenge as there are so many ways to score in VEX; fortunately, our design narrows our possibilities to the (relatively) simple task of scoring rings on mogos.

Our ABP Goals

Ideally, to ensure the ABP, we should attempt to **score at least 3 rings** on mogos, securing a total of 6 points (3 for rings, 2 for top ring). This is especially achievable as there are at least 3 rings within (relatively) easy reach of the starting line.

AWP

Overview & Our AWP Goals

AWP is a more defined goal, although considerably harder to achieve; this is becasue the manual specifies the goals to achieve – for AWP (at regionals) we need to:

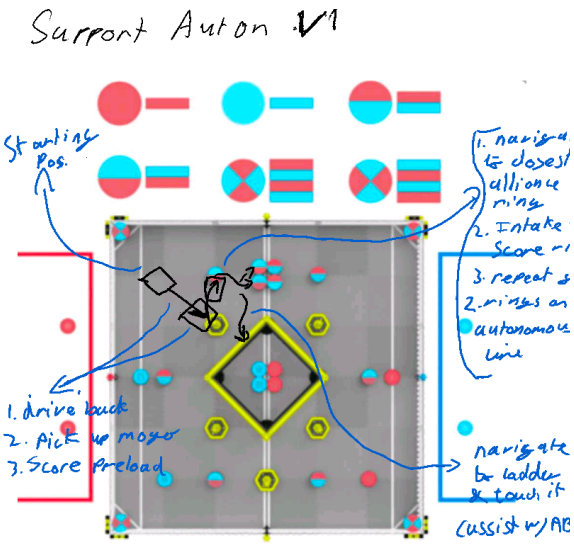
1. At least 3 rings scored
2. A minimum of *2* stakes with at least *1* ring scored on them
3. Neither alliance contacting/breaking the plane of the starting line
4. At least 1 robot contacting the ladder (at the end)

[1]

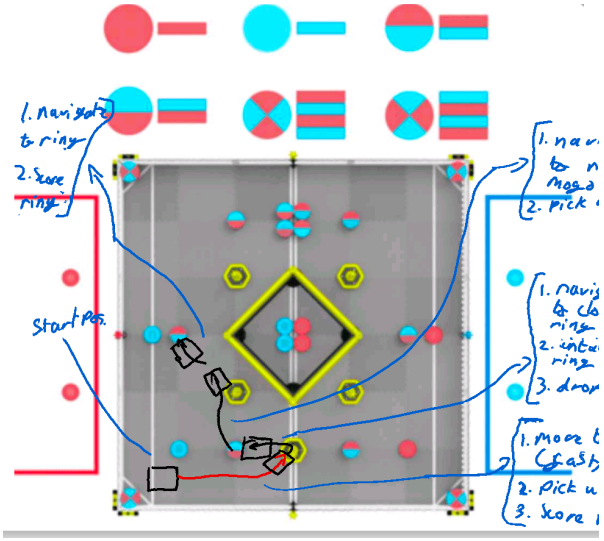
Planning Route

I chose to first draw out a plan for what we would like our autonomous routine to acheive and how; they are heavily inspired by the any spectactular autons that already exist and have been showcased at MOA or Lobstah Bowl.

[1] Autonomous Bonus Point
[2] Autonomous Win Point(s)



The sketched plan for a 'support' (ABP Securing) auton



The sketched plan for the AWP 'rush' (grabs far mogo) auton

Using these plans, I can get a sense of what sort of movements will be required.

Coding Support Auton

The 'support' auton is an auton that is used to support an alliance in securing the ABP, this effectively takes shape as scoring as many rings as possible, from the side where the negative corners are placed.

The Code

The code can be seen as followed:

```cpp
void Autons::support() {
    Robot::Pneumatics::mogoMech.retract(); // make sure mogo is not clamped.
    Robot::Pneumatics::intakeLifter.retract(); // make sure intake lifter is not lifted.
    Robot::Pneumatics::doinker.retract(); // make sure doinker is not out.
    if (!pros::competition::is_autonomous()) calibrate(); // calibrate if testing (not comp)

    chassis->setPose({-50, 41, 305}); // set initial pose
    // move to mogo and pick up
    chassis->moveToPoint(-32, 28, 1000, {.forwards=false, .maxSpeed=110, .minSpeed=40});
    chassis->waitUntilDone();
    Robot::Pneumatics::mogoMech.set_value(true);
    delay(200);
    // move towads ring while intaking
    chassis->turnToPoint(-24, 48, 500);
    Robot::Actions::setIntake(1, Types::BOTH);
    chassis->moveToPoint(-24.5, 46, 1000, {.maxSpeed=90, .minSpeed=40});
    // move back slightly to help with intaking
    chassis->moveToPoint(-23, 44, 750, {.forwards=false});
    chassis->waitUntilDone();
    delay(500);
    // turn to, and navigate towards rings on auton line
    chassis->turnToPoint(-3, 46.5, 800);
    delay(500);
    chassis->moveToPose(-2, 47.5, 117, 1000, {.lead=0.5, .maxSpeed=80});
    chassis->waitUntilDone();
    // wait to suck up first ring
    delay(500);
    // suck up second ring
    chassis->moveToPose(-24, 48, 140, 2000, {.forwards=false, .minSpeed=40});
    chassis->turnToPoint(0, 40, 750);
    chassis->moveToPoint(-9, 39, 1000, {.maxSpeed=65, .minSpeed=30});
    chassis->waitUntilDone();
    // wait to suck up
    delay(500);
    // move away
    chassis->moveToPose(-10, 48, 180, 1000, {.forwards=false});
    // move towards ladder (helps w/ AWP)
    chassis->moveToPose(-24, 24, 170, 3000);
    Robot::Actions::runIntakeFor(1, BOTH, 2500, true);
    chassis->waitUntilDone();
    return;
}
```

Commented code that performs support auton route

Coding Rush Auton

The 'rush' auton is an auton that is used attempt to secure AWP without relying on the alliance partner[1], it would envolve scoring 3 rings on 2 stakes and touching the ladder.

The Code

The code can be seen as followed:

[1]They would still have to move off autonomous line.

```cpp
void Autons::support() {
  Robot::Pneumatics::mogoMech.retract(); // make sure mogo is not clamped.
  Robot::Pneumatics::intakeLifter.retract(); // make sure intake lifter is not lifted.
  Robot::Pneumatics::doinker.retract(); // make sure doinker is not out.
  if (!pros::competition::is_autonomous()) calibrate(); // calibrate if testing (not comp)

  chassis->setPose({-50, 41, 305}); // set initial pose
  // move to mogo and pick up
  chassis->moveToPoint(-32, 28, 1000, {.forwards=false, .maxSpeed=110, .minSpeed=40});
  chassis->waitUntilDone();
  Robot::Pneumatics::mogoMech.set_value(true);
  delay(200);
  // move towads ring while intaking
  chassis->turnToPoint(-24, 48, 500);
  Robot::Actions::setIntake(1, Types::BOTH);
  chassis->moveToPoint(-24.5, 46, 1000, {.maxSpeed=90, .minSpeed=40});
  // move back slightly to help with intaking
  chassis->moveToPoint(-23, 44, 750, {.forwards=false});
  chassis->waitUntilDone();
  delay(500);
  // turn to, and navigate towards rings on auton line
  chassis->turnToPoint(-3, 46.5, 800);
  delay(500);
  chassis->moveToPose(-2, 47.5, 117, 1000, {.lead=0.5, .maxSpeed=80});
  chassis->waitUntilDone();
  // wait to suck up first ring
  delay(500);
  // suck up second ring
  chassis->moveToPose(-24, 48, 140, 2000, {.forwards=false, .minSpeed=40});
  chassis->turnToPoint(0, 40, 750);
  chassis->moveToPoint(-9, 39, 1000, {.maxSpeed=65, .minSpeed=30});
  chassis->waitUntilDone();
  // wait to suck up
  delay(500);
  // move away
  chassis->moveToPose(-10, 48, 180, 1000, {.forwards=false});
  // move towards ladder (helps w/ AWP)
  chassis->moveToPose(-24, 24, 170, 3000);
  Robot::Actions::runIntakeFor(1, BOTH, 2500, true);
  chassis->waitUntilDone();
  return;
}
```

Commented code that performs support auton route

Getting Ready For JWS 2024/09/08

On the 28th of September we are attending the regional competition at The John Warner School in Hoddeston, as with any competition we cannot take the whole workshop with us, we must travel light and pack efficiently.

Packing for JWS

Before the competition begins, we must ensure that we bring everything we need to compete, while also keeping spare parts for the sake of redundancy. Shown below is the list of things we must bring with us:

- Robot (obviously)
- Battery X2 (with charger)
- Screws
- Nuts
- Long C Channel (for quick replacements)
- All pistons (in case we have an unexpected malfunction)
- Pit decorations
- Saw (quick replacements/adherences, that require cuts)
- Rubber bands
- Laptop (last minute code adjustments)
- Licence plates (red and blue)
- Fan (for cooling overheated motors)
- Zipties
- Controller
- Pneumatic pump (with charger)
- Safety Glasses (per S1 [1])
- Drill

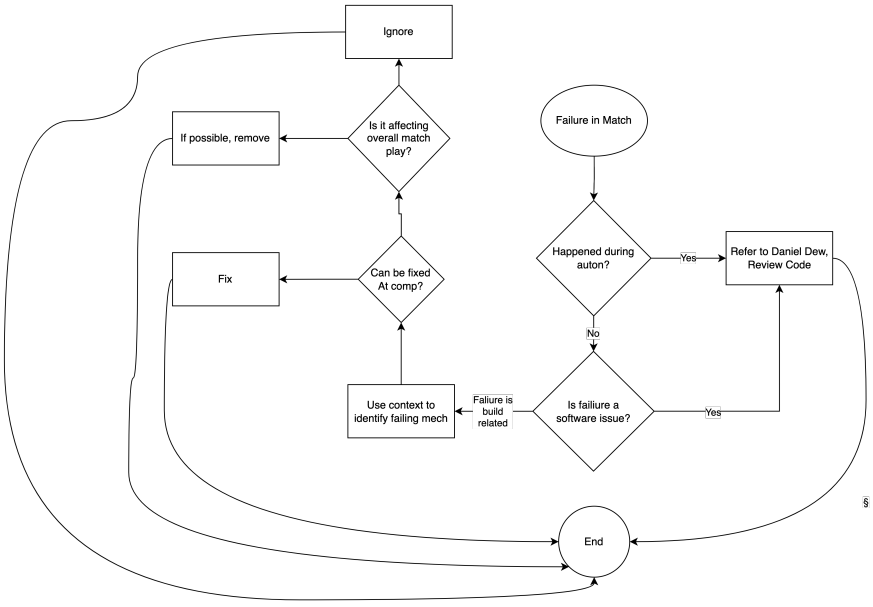
> **Quote**
>
> “If you need one, you bring two “

Crisis Management at JWS

While this seems like a dramatic caption, we need to define clear and easy to fillow guidlines for if something goes wrong.

Faliure During Match

Handling a faliure (non-driving) during a match can be handled like this:



A flowchart depicting how faliures during a match can be handled

Reconaissance at JWS

Usually, before a competition we like to refer to our notes and social media to see not only which teams and robots would be possible threats, but also which teams we would want on our alliance. Unfortunately , because this is the first competition of the season, we have very limited information (if any!) about the other teams, so we have to go into the competition “blind” and rely solely on our scouting at the beginning, adn throughout the competition.

As head of strategy, it is my role to go round to all the other teams at the start of the competition and make notes on a variety of things, such as, but not limited to:

- **Autonomous**
  - Do they have solo AWP Autonomous?
  - Can they do an Autonomous from both sides or just the one?
  - Do they have multiple Autonomouses and if so which one works best with our robot?
  - Is the Autonomous reliable?
  - Do they even have an Automnomous??
- **Robot limitations**
  - Can they score on Neutral Wall Stakes?
  - What level can the robot climb to? (if it can climb at all)

  - Can the Robot descore?
  - Can it even possess rings / MOGOS?
- **General build quality**
  - Is it a fast robot?
  - Does it break / fall apart easily?
  - Is the intake reliable?
- **Does it work well with our robot?**
  - Does the Autonomous intefere with ours?
  - Can they score on the neutral stakes if we can't?

All these factors play a massive part in helping us formulate a strategy for a game to game basis, for example, if we know that both opponents have an auton that only works on one side, we can confidently do a slightly riskier autonomous on the other side, as we know there is less chance of an entanglement / interaction at the Autonomous line, and less chnance of getting a violation in rules such as G11 and G12. Controversially, these facts also help us in choosing an Alliance partner (if we get there), as we can decide on a robot that works nicely with our robot, for example having a support Autonomous on the other side, and are a team, who we believe, will help us in (hopefully!) becoming Tournament Champions.

At the pit checklist

1. Motors are plugged In
2. Sensors are plugged In
3. All breaks are repaired from the last Match
4. All screws must be tight
5. Pneumatic pump is locatable
6. Check length of skills queue
7. Colour of next Alliance

At the field checklist

1. Safety Glasses
2. Pneumatic pumped
3. Battery
4. Controller
5. Drive-team members
6. Robot drive is working
7. Robot intake is working
8. Plates are on and correct

Introduction

After discussing the results of the JWS Regional Competition in person, we have decided to compile our thoughts into a chapter as an evaluative exercise and in order to keep us on track with the goals we outlined at the start of the season in the chapter: Managing the Team. We believe that doing a review in this way will help us improve and become better for next time.

The Competition

Qualification

In the JWS Regional, they cancelled our practice match because they were runnning behind schedule, which lead to our first ever draw in a VEX match as for Q2 we forgot to fill our pneumatics, which meant that we couldn't possess mogos or score rings onto it. From this we can takeaway that VEX competitions are chaotic and its difficult to predict what is going to happen therefore its important we are as careful as possible to avoid making minor mistakes as they can have match affecting results. For the rest of the competition, we were extremely cautious about our tanks and made sure they were always filled every match. By accounting for everything that we can control, we will in future greatly reduce our chance of failure.

Our second qualifying match - Q11, we played against 10478S Gearers and 13765T MTS_Bubblegum, whilst being allied to 10478B SubZeroRobotics. We lost this match 19-3 because we let them take positive corner first, which gave them a sizable point advantage. However, we managed to win our 5 other matches, winning AWP for 1 of them. In a later chapter our head strategist, Thomas Robb, will provide an indepth analysis of strategy and plays made during qualification and elimination. This meant that we qualified 4th out of 24 teams. Looking at our objectives, this means that we have definitely succeeded with our first objective "A Force to Be Reckoned With".

Aliance Selection and Elimination

For Alliance Selection, 17711E Entropy were our alliance partner, meaning we made No. 1 Seed for elimination. We picked each other because we were both consistent bots, both with the ability to carry the match if needed to. We won our QF match against 1875N Noodle and 13765T MTS_Bubblegum by securing positive corners.

For the SF we were up against 15650C Memento Mori and 10478S Gearers, in this match both Red and Blue alliances were disqualified because of major violations, that were both match affecting, we were called up on ploughing a mogo whilst possessing one already. This meant that our match had to be replayed. However this time both 15650C and 10478S were able to secure positive corners and auton bonus additionally, which meant that we lost 13-28. In future, our drive team are looking to ways to avoid this as much as possible as match replays cause shaken confidence and can lead to losing critical matches, which in this case it unfortunately did.

Skills

Since we didn't have enough time to perfect our autonomous skills before this competition, we didn't submit any autonomous skills runs, which disqualifies us from recieving the Excellence award. We ended doing one drivers skills run, where we obtained 17 points, which placed us 8th out of 16 on the skills leaderboard.

Interview

Overall, we believe that the interview went fairly well. We discussed practically every aspect of our robots design with the judges and also everyone contributed the conversation at several places throughout the interview.

Notes From Team

It is sometimes very useful to note down what each person initially thinks, as often the 'face value' notes are the most useful to keep in mind.

Daniel Dew

Personally, I think the competition was a success, not only becasue we brought home the Judges Award, but also because of the key practice and experience we gained. As the programmer and driver, High Stakes brings several complex problems that seem huge and rather difficult, JWS helped to solve some of those issues and put the game into perspective – a much needed boost of confidence.

Jonah

As a member of the drive team and a builder, the competition and promise we showed at JWS really boosted my confidence in what we can achieve in the future. I feel that, since last season, we have improved as a team much more than we realized. Our strategy and driving set us apart as one of the top teams in the UK, even when they didn't always align perfectly with our robot's capabilities. <<<<<<< Updated upstream

Daniel da Silva

In my opinion, I'm very happy with our debut to this season. After a slightly rocky start, we were able to recover and not lose a match until the Semi-Finals, despite this I think the drive team did excellent in the face of difficult opposition. I'm very excited to see how much we improve from here.

Daniel DaSilva

>>>>>>> Stashed changes

Thomas

In such an unpredictable game like high stakes, coming 4th was a success for us, and we were delighted to alliance with Entropy, which is in my opinion, one of the most powerful teams out there. Despite losing the semi-finals through a very close replayed match, we were thrilled to have not only come away with a judges award, but also a (hopefully!) strengthened bond with Entropy, as,despite losing, our teams worked well together with not only great robots, but great drivers too. Looking back, we have definitely noticed flaws and mistakes in both the robot, and also our game strategy, which will both be duly implemented in time for our next competition. We enjoyed being back competing, and teams will definitely be seeing us again.

Aubert

As a builder, I thought this was a very important competitiom, it allowed us to get a strong grip on the nuances of this competition in terms of strategy, programming and building. It also allowed us to gain more insight onto what building for this season will entail – for example, the stress requirements of the drivetrain and surrounding subsystems that can't necessarily be discovered during practice alone.

Results and Reflection

Overall for our opening competition of the season, we are on the whole quite proud of what we achieved in this competition. We learnt valuable strategic information such as the importance of having an organised pit team and having the robot fully ready before a match. In addition, we played generally quite well winning most of our qualifiers, ranking 4th and getting picked by 1st seed. Additionally, we were awarded the Judges Award as special recognition for our ability and our interview performance.
In terms of how this competition puts us in relation to our goals, we are still on track for "A Force to be Reckoned With" making first seed this competition but we did not win an award that qualifies us to nationals at this competition. Winning a worlds qualifying award is not possible at this stage, because there aren't any Signature Events in the UK and none are reasonably close enough to fit into our budget or school routines given we are Year 12 team (equivalent to Junior Year in US). This means that Nationals is a much more important goal to work towards than Worlds currently.

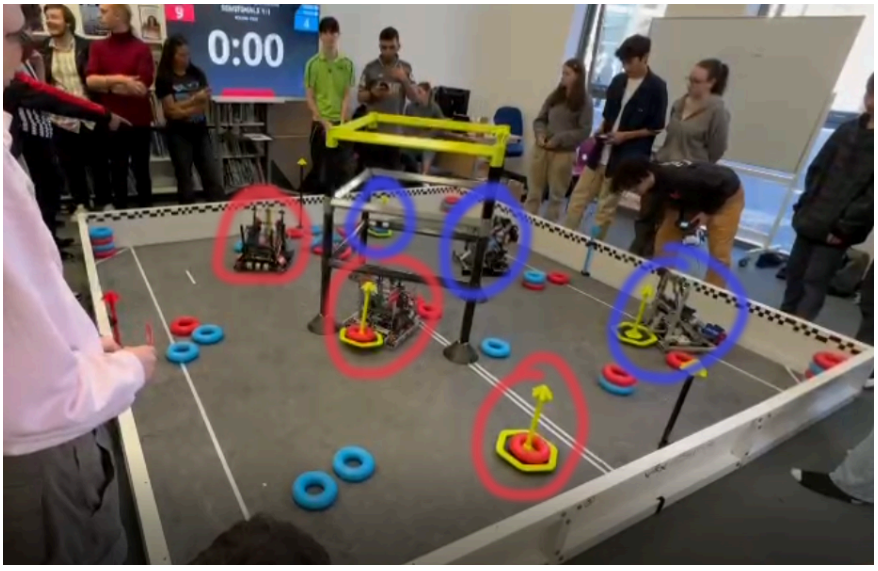
JWS analysis

Despite losing more than 1 game in this competition, I will only be focusing on, what I belive to be, the most strategically interesting game out of all the ones we played, which was our 1st Semifinal match, which ended in a double disqualification. I will be looking at what we, and all the other teams did, and, looking back on the game, what we, and our alliance partner, Entropy should have done differently, to avoid our disqualification, and to put us in a better position overall.

Semifinals 1-1 (Double disqualification)

Autonomous

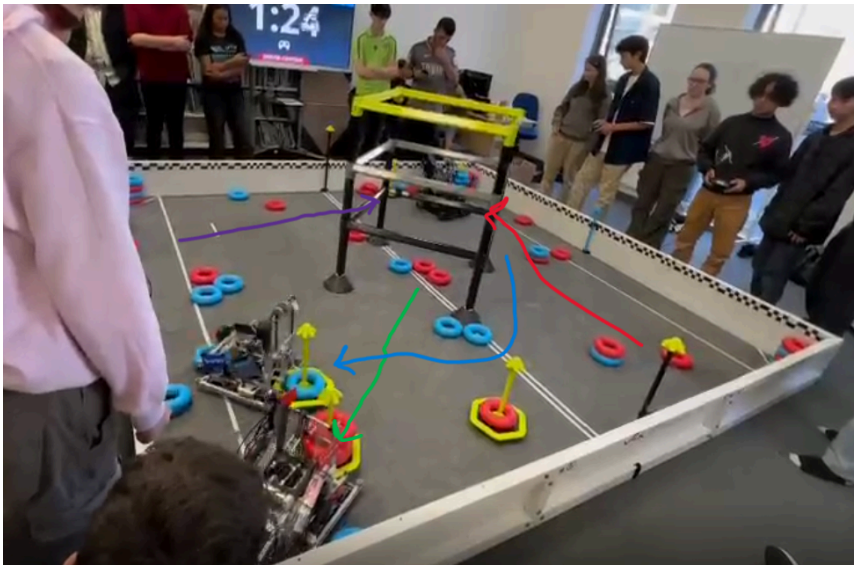
Since the auton is all pre-coded, there is little to no strategy that can be applied during the autonomous period, only what the next course of action should be based on the other teams positions.



Here, you can see our robot has got “base control” of 2 MOGOS , having scored one red ring on both, which, despite being 1 less than our auton can do if done perfectly, is a good controlling start to the game. We can also see Entropy (our alliance member) controlling and possessing their own mobile goal, with a similar 1 red ring being scored.(All highlighted in red)

One of the teams on the opposing alliance, Memento Mori, managed to get an impressive 2 blue rings on their MOGO,(highlighted in blue) while their fellow alliance member, Gearers, didnt have an auton, and subsequently didn’t move off the starting line throughout the period. This meant that, at this point in the game, we have more control over the field, but as you will soon see, this will quickly change.

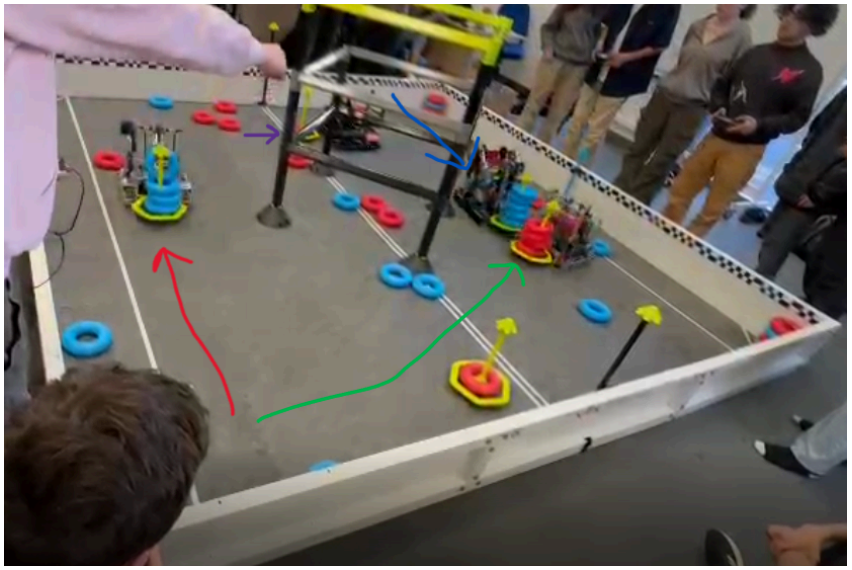
Driver Control



As soon as the Autonomous period ends, we (represented with the green line), move towards the positive corner, in order to gain the 2 red rings while our teamate, Entropy, make their way towards the negative corner, in order to try gain their mobile goal back, as they have temporarily dropped it, most likely trying to score the wall stake, before changing their mind.

Looking back at this moment, I believe that we both made mistakes, as we should have perhaps gone for the unguarded centre rings, realsing that Gearers had no way to clear the corner, and therefore cannot place a mobile goal as per .

In my opinion, Entropy should have never dropped their mobile goal, and, although we dont know if it was a mistake or purposeful, it is a flaw in our Alliances game strategy.

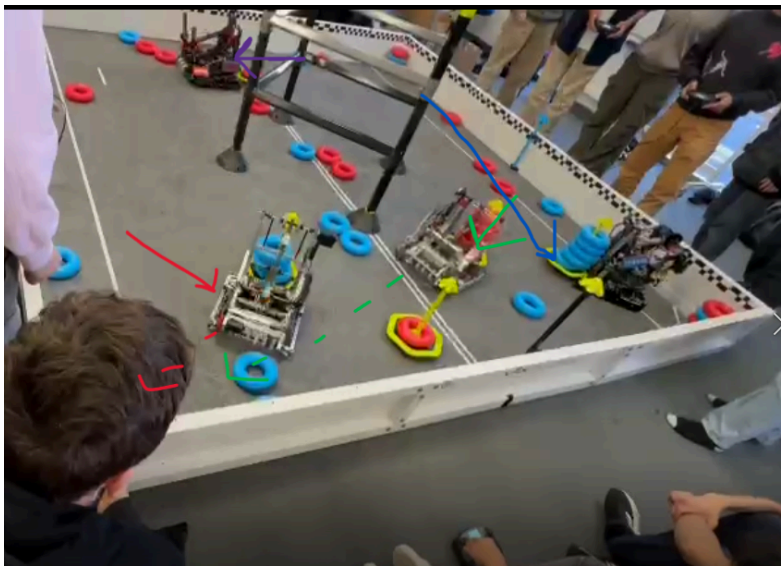


After successfully filling up our MOGO with 4 red rings, we decide to head to the Blue Alliance Side, in order to gain more red rings. We encouter Memento Mori, and attempt to block them from getting to the corner. However, our robot was not robust enough at this

stage, so we were unable to, and they unfortunately managed to clear, and place their MOGO in the corner. Entropy is continuing to try get mroe rings on their Possessed MOGO, but are unable to. At this point Gearers are also close to a full MOGO, and we observe teh clear corner next to them.

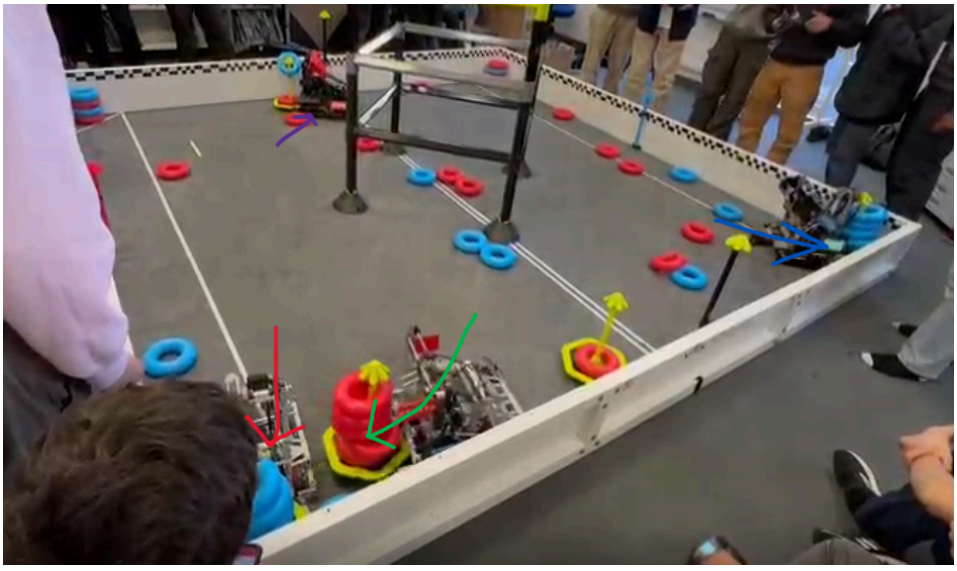
Reflecting on the game, we should have signaled to Entropy to come and block the right postive corner, while we should have gone to the left postive corner, as we had a much fuller MOGO than Entropy.

We also took note that we would need a more robust and stronger robot for future interactions like this, and this note was factored into our design proccess for our upcoming rebuild.



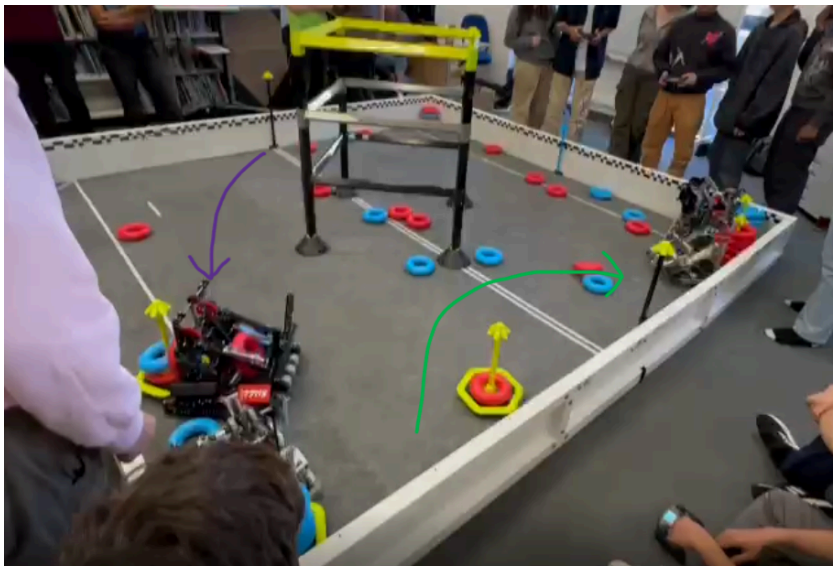
Realising (a bit too late) that Gearers are trying to place a goal in the postive corner, we quickly begin to move towards the repective corner, while gearers are trying to fully fill up their MOGO. It is also important to note that Memento Mori are about to clear their corner to, likewise, score.

I believe that we played this part well given the postion we were in, as we pre-emptively saw Gearers trying to score, and attempted to stop it. My only criticism would be to try and communicate to Entropy more, and hopefully get them down this end of the field, where the majority of scoring was about to happen.



Memento Mori successfully manage to score their MOGO in their corner, while Gearers manage to do the same.

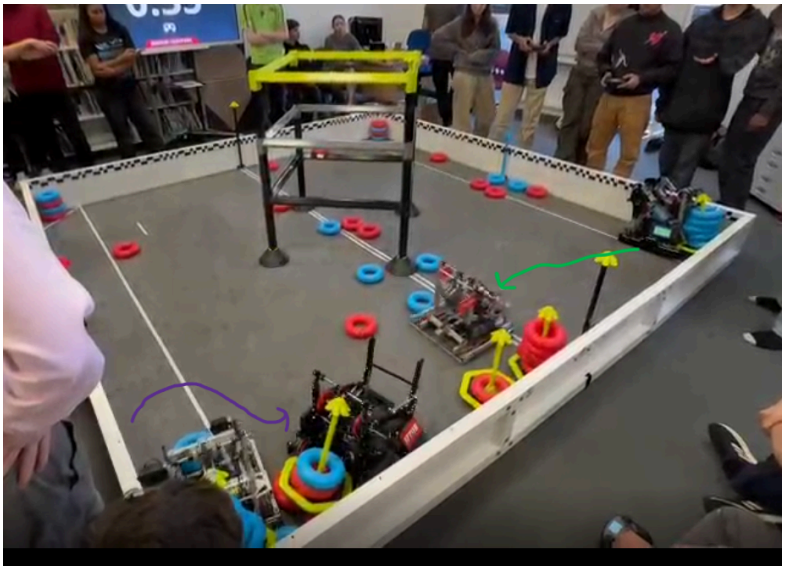
In Hindsight, we should have come to the corner a little earlier, and then we could have perhaps bloakced Gearers, or possibly place our own mobile goal. Once, again we should have communicated to Entropy more, and get them to come down to the positive corner end of the field.



Entropy finally makes their way down to the positive corner, where they have accidentally scored a blue ring on their mobile goal, giving the Blue alliance more points on that goal, at 3-1. We try to push ouir goal into the corner with Memento Mori, but once again, with our first robot version, are unable to do so.

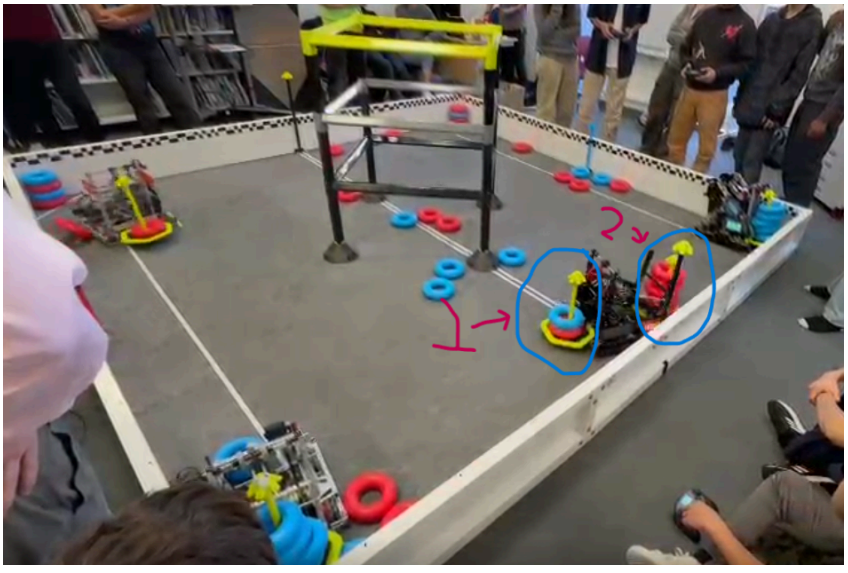
At this point the Blue alliance have a huge lead on us, with many more rings scored than us, the majority being doubled becasue of positive corners. Ideally we should have prevented this

from ever happening, as this puts us in an incredibly difficult position, but now we just have to try and regain some control. Unfortunately, since neither of our robots can climb, it is impossible for us to win unless we manage to remove one of the Blue Alliances goals from the corners. So, we have to play offensive.



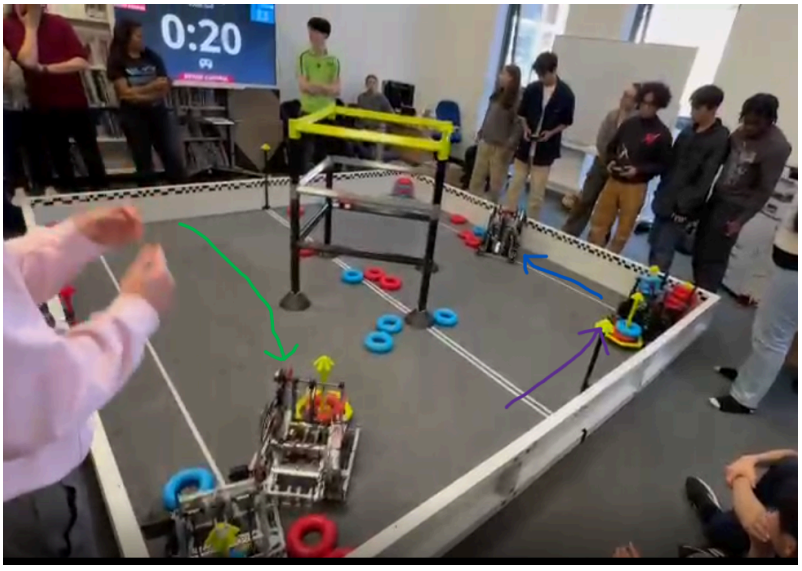
As Gearers robot is less robust, and easier to move, Entropy begins to try and push them out of the corner, in order to allow us to place the full MOGO in teh positive corner. However, we quickly realise that this doesn't work due to the angles Gearers are at, which makes them very difficult to move. We decide to pivot to try and remove Memento Mori from the corner, and, as Entropy has the stronger robot, we drop our MOGO to allow them to score more points, while we try to fill up the 5th mobile goal, which has not been used for the Driver control period at all.

I believe this is the correct move to make given the circumstances, and, although we were in a tough situation, we persevered and came up with a good, potentially game winning strategy.



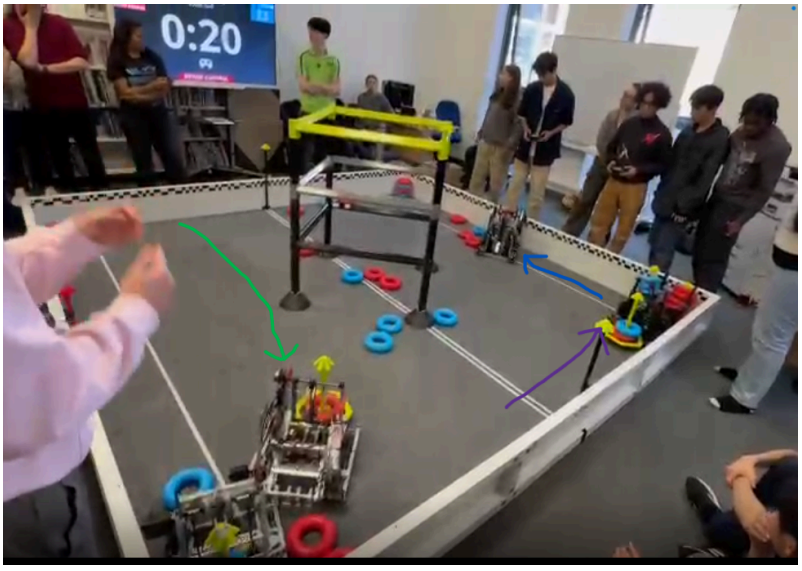
Despite our good strategy, this is where it all goes wrong. As you can see from the picture below, Entropy is contacting 2 mobile goals, and they are Possessing MOGO 1, and Plowing MOGO 2(As illustrated). This becomes a direct violation of and since they end up plowing this goal into a corner, it is egregious and clearly intentional, making it match affecting and therfore a major violations, leading to a disqualification.

There is nothing we could have done in this situation, and it is ultimately Entropy's mistake, and, in their defense, it was an easy one to make, which unfortunately led us to being disqulified. Keep in mind we did not know for sure if we had been disqulified, as the referees may rule it differently so we continued to play at our best and the match was far from over.



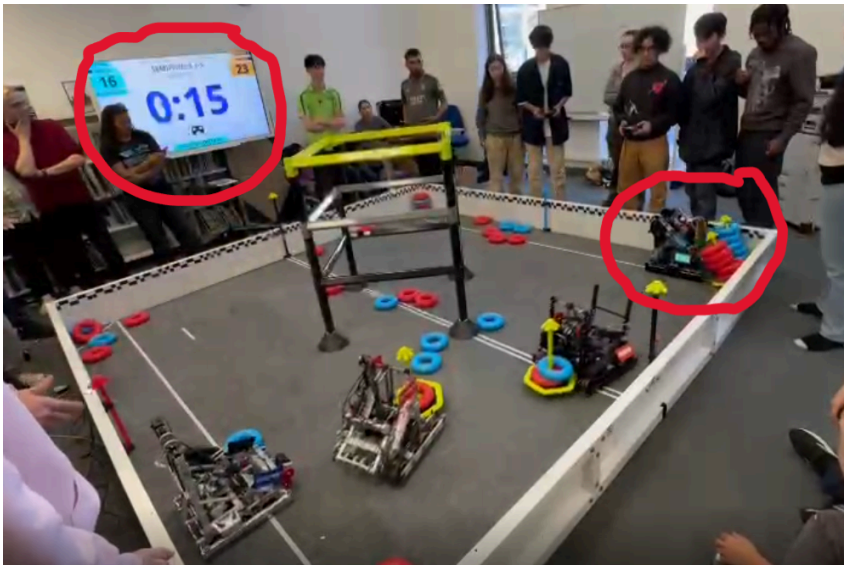
Entropy manages to place the mobile goal in the corner, and push Memento Mori's goal out of it. Memento Mori could not fight back, due to entropys position, and end up trying to score a wall stake instead. We make a last ditch effort to try and place our mobile goal in the corner which Gearers is currently protecting, but are unable to do so.

It is important to notice the time (20 seconds left), as per , positive corners are protected in the last 15 seconds. I think this was a brilliant play from our alliance, especially Entropy, who managed to turn the tides on this once unwinnable game, and possibly allow us to qualify to the finals.



As the time hit 15 second, Us, Entropy and Gearers quickly moved away from the corners, but Memento Mori continue to try and push their mobile goal into the corner, and manage to, and also push ours out. If this had only been a few seconds earlier, this would have been a brilliant, and game changing play. However, as you can clearly see from the image, this was done in the last 15 seconds, and this causes Memento Mori to be in direct violation of , casuing them to get a major violation, and hence a disqualification.

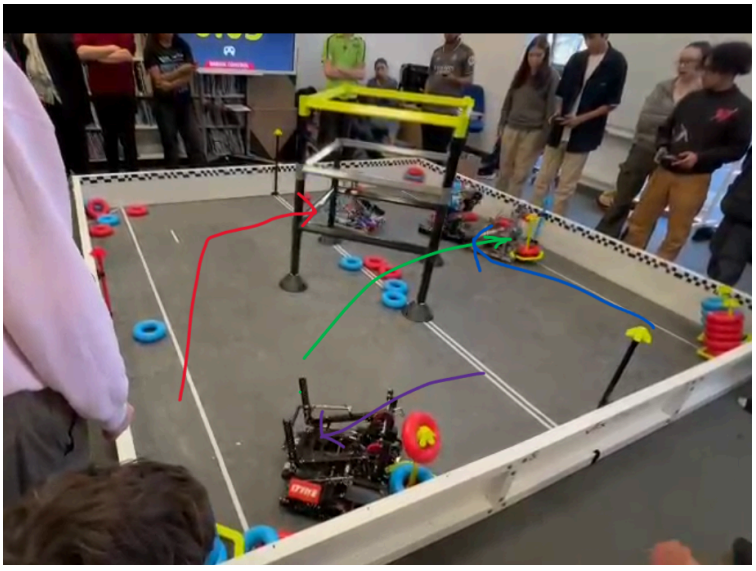
This was an incredibly lucky moment for our alliance, as Memento Mori were under the impression that they were going to lose, however the score at this point was actually 22-18 to them, so if they didn't make this move, they still would be winning. Furthermore, they had not realised that we were disqualified, and if they knew this, they would have most likely made less risky plays, and not cause themselves to be disqualified too(this is why you need a strategist on your team!).



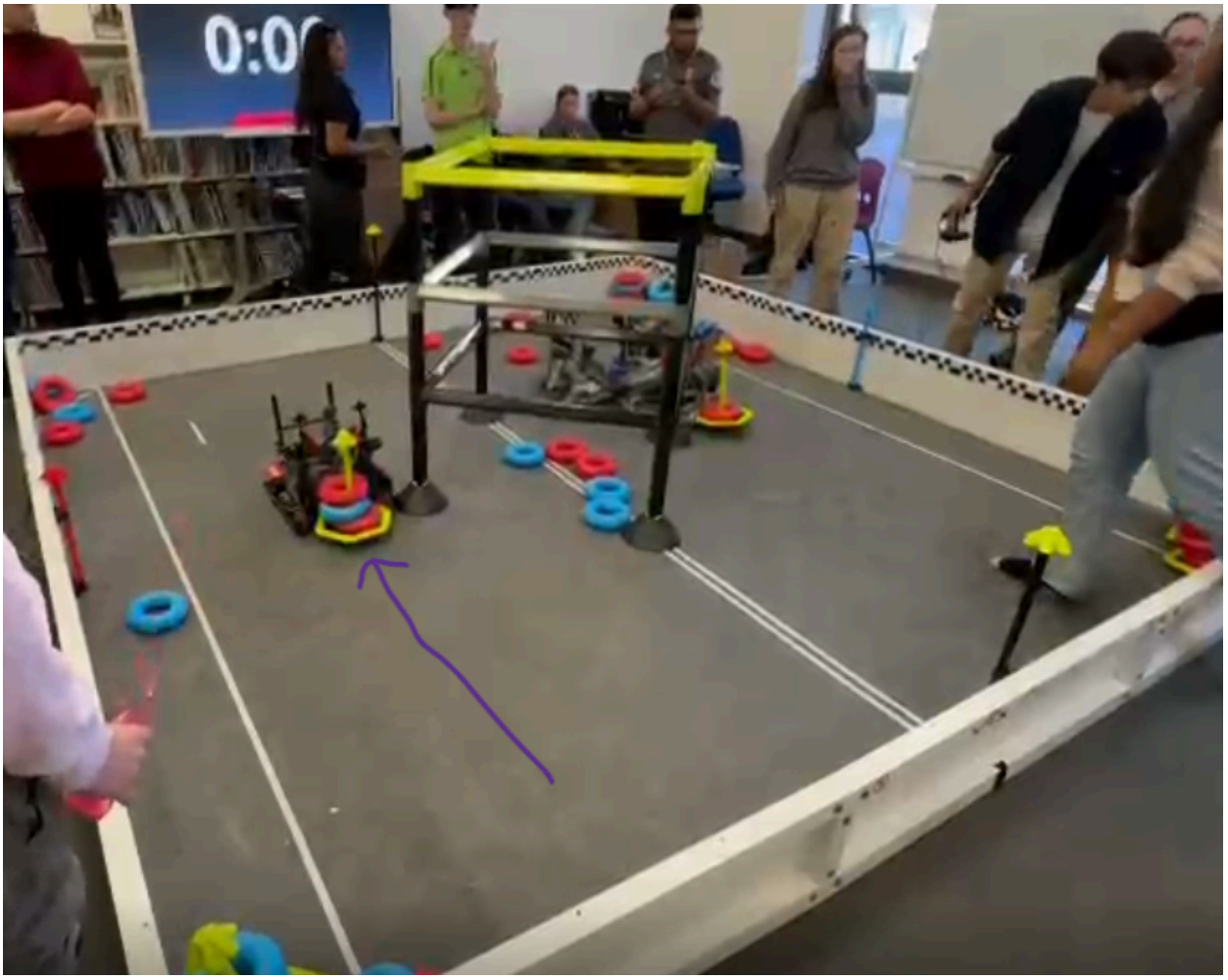
Screenshot 2025-01-02 155140

Knowing Memento Mori would most likely be disqualified, we played the rest of the game safely, with Entropy cleverly scoring a red ring on top of the blue one, causing the MOGO's points to switch from 1-3 (red-blue) to 4-1. Since none of the robots could climb, and 3/5 MOGOs were placed in corners, Memento Mori and Gearers just tried to stop us from scoring anything, adn towards the end of the game, Memento Mori tried to score a wall stake, but ran out of time.

With nothing more we could do, we decided to play safe (in case we weren't already disqualified), and try to score a few more rings in the last few second, but were blocked by Memento Mori and Gearers.



This image marks the end of the game, and you can clearly see that Entropy managed to get one more ring on, as well as us being blocked by Gearers and Memento Mori.



Conclusion

In summary, it is very difficult to get a strategically perfect game, as when you're actually on the field and playing, it becomes an extremely stressfull enviorenment, particularly for the driver, who is responsible for making tough, split-second decisions all the time, and that 1:30 can feel so quick.

Saying this, I believe that no team played at their strategic best, very much including us, and from this match there are a lot of mistakes, as well as successes, btgo with strategy and robot design, that we can take away from this experience, and improve by, in order to become stronger and a much more threatening, as well as careful, team for our future competitions.

Designed by: Snowflakes
Witnessed by: Daniel da Silva

? Problems Presented at JWS 2024/09/30

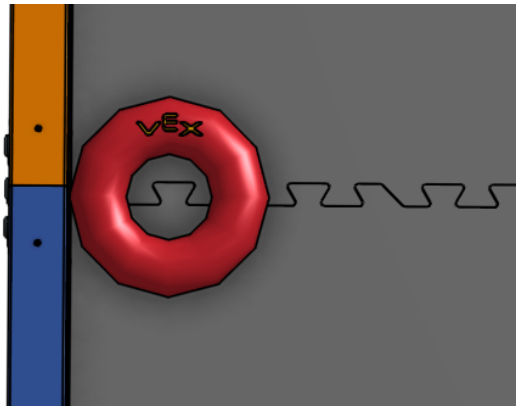
After a succesful competition at the JWS regional there is much to reflect on about the performance of our robot, as we now have true match data showing how we can perform in a competition environment.

Improvements:

Although we were among the top contenders in terms of competition at JWS, forming the first seeded alliance, and making it to semifinals through some arduous matches, it is clear that we need to improve several aspects of our robot's design.

The intake:

In the first iteration of our robot, we used a flex wheel intake with three separate flex wheels driven by a direct 600 RPM motor cartridge at 11W. Due to the limited number of flex wheels available, this intake mechanism did not provide a strong enough grip on the ring to achieve the desired intake performance. During matches, we noticed that the intake often struggled to pick up rings unless they were pressed against the wall, as shown below:



Ring pressed against the field barrier, allowing smooth transition due to force applied against the intake rollers

Our goal is to intake rings without needing to drive over them, which will improve the consistency of autonomous routines and provide smoother gameplay by eliminating the pauses caused by the inconsistencies of the first iteration.

Scoring mechanism:

For JWS, we used a 'hook' design to transport rings up a chain pathway and pivot them at high speed onto the mobile goal, effectively addressing the challenge posed by the flange at the top of the goal. However, this design had limitations in terms of speed. To maintain scoring consistency, we had to run the chain relatively slowly, which led to long cycle times. The downside is that spending extended time on each mobile goal and ring gives our opponents the opportunity to secure the positive corner. One lesson learned from JWS is that once both positive corners are lost, it is almost impossible to win.

Mogo Mech:

In iteration 1 of our robot, we used a simple diagonal clamp to secure mobile goals. While this approach was lightweight and straightforward, it wasn't very secure and added unwanted compression to certain parts of the drive, particularly the braces. When the mobile goal mechanism was activated, it caused a squeezing effect between the drivetrain's two piston contact points, which negatively impacted bracing; ideally, we want all components to maintain right angles, with no bends in the C-channels. We also experienced one instance where the mechanism was stolen, which was less than ideal and ultimately cost us a match.

Drivetrain:

Iteration 1 featured a 450 RPM, 2.75-inch, 8-wheeled drivetrain that performed reliably throughout the competition, leaving no areas for improvement. The hot-swappable motors enabled us to cool the drivetrain quickly and efficiently, while the 450 RPM allowed us to be the fastest on the field most of the time. Our bracing ensured the drive remained stable and frictionless, even under intense stress. We plan to continue with this drivetrain in future iterations.

In conclusion:

The elements of our robot with serious flaws / needing improvement are:
- Intake (grip on ring is weak and exterior forces are required for consistency)
- Scoring mechanism (slow and inconsistent leading to slow cycle times and mis-scoring)
- Mogo Mech (easy to steal from, flimsy and bent parts of the drivetrain)

Decision to rebuild:

After analysing each part of our robot that required improvement it is obvious what the next step for our robot's development is: a rebuild of all mechanisms **apart from** the drivetrain. This will allow us to move forward into competitions with a higher chance of winning and securing a place in nationals. Our rebuild will involve improved versions of all of the above.

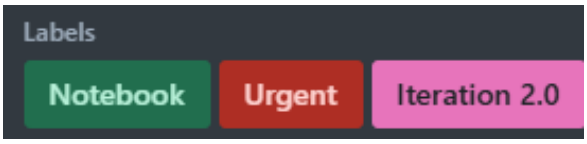
Leading the team/rebuild (Daniel Dew):

With such a tight timeline, as we must be ready for the Coventry Regionals, it is extremely important that we maintain high standards, therefore team management must be used effectively to achieve maximum results.

Keeping On Top of Tasks

With a rebuild, comes many different aspects to keep on top of on including:
- CADing the new iteration
- Building the new iteration
- Logging/notebooking on changes
- Programming – suitable code, auton routines etc
- Driver Practice – including skills

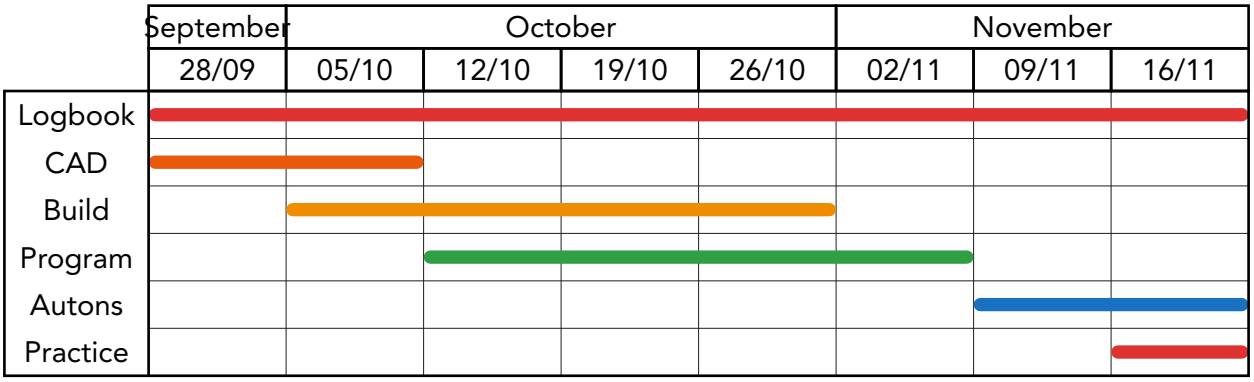
To manage this huge new influx, we will be heavily using Trello, the task management software mentioned before: we can label each new task with a new label, indicating the new iteration, urgent tasks can also be labeled – and as always the type of task is indicated (example below).



A notebook task labeled as urgent and a part of 'Iteration 2.0'

Planning the Rebuild – Time

Another huge aspect of managing the team through a rebuild is time management, below I have outlined (in a Gantt Chart) the timeline of this new design cycle.



Note

Notes on Gantt Chart

- Logbook
  - Logbook is done throughout, and kept up to date based on developments
- CAD
  - CAD for base features aimed to be finished by 05/10
  - Complex system CAD aimed to be finished by 12/10
  - Building of core features can overlap with CADing of complex features
- Build
  - Marked timings is for newer or more complex features, therefore simple changes (< 2hrs work) can be done whenever
  - Overlap with CAD as mentioned above
- Programming:
  - Programming can be developed as building requires (e.g, new feature built, feature is then implemented in program)
  - Supplementry features can be developed whenever
- Autons
  - Tuning of control system parameters must not take place until robot is >90% finished, to avoid weight changes affecting control systems.
  - We must aim for at least 2 match autons (different sides) and a programming skills route.
- Practice
  - Drills, friendly matches, and familiarisation.

Deciding on Improvements for V2 2024/10/05

The interest of time

In order to compete as effectively as we can we want to improve our overall performance as much as possible. The three constiuent parts that affect this broadly fall under one or more of these 3 categories:

- robot design
- autonomous routines
- driver skill.

Ultimately, effective time management of these three aspects is vital for robotic success within the world of VEX. It is also important to note that we can not overcommit ourselves to VEX or work inconsistently on this project in order not to overburden the team and cause social and academic burnout.

Additionally, we must consider the order in which we implement changes noting that rebuilding tasks can generally not be done in parallel with driver practice or autonomous. Also building should be done before programming because its important that for as much time as possible we have a functional robot within the scope of the game's meta because that allows for more practice time and also more time to tune auton and secondly it is taken as a given that any changes in terms of drivers practice or autonomous routine will be renderred useless because of the many micro adjustments in a rebuild.

That all being said we are most likely to approach the time before the upcoming competition by firstly recadding/building then improving autonomous and getting in drivers practice simultaneously. Below we shall discuss what we have planned to change in terms of our robot's design based on performance in the most recent competition (as of time of writing), which was the JWS regional.

Robot Analysis

Before we can make any changes we need to decide what needs to be changed. We have concluded that there's nothing wrong with our drive base since we took a lot of care into getting that right the first time and we found that our drive worked well for us in matches. However, in order to make room for the changes, which we want to do to the bot we will almost certainly need to redo the drive bracing.

One of the things which is being reworked is the mogo mech: we are going for the same dual piston design architecture but with a vertical layout rather than a pivot based design. This will **drastically** change the angle at which the mogo is clamped.

However, this ties into the changes we are planning to make to our intake and ring mech. We are planning to retain our current first stage design but we want to lower the profile of the mech. We are also considering implementing a wall stakes mechanism in our second iteration.

Below are the changes that we have considered and reevaluated in terms of existing systems.

Differential vs Holonomic

	Manoeuvrability	Stability	Complexity	Wheel Size	Gear ratio achievable	Total
Differential	3	5	5	4	5	22
Holonomic	4	4	2	4	2	16

Looking again at the number of wheel's used

	Manoeuvrability	Traction	Complexity	Gear ratio achievable	Total
4 Wheels	5	2	5	3	15
6 Wheels	4	3	4	3	14

8 Wheels	3	5	3	5	16

Reconsidering the types of wheels which we use.

	Manoeuvrability	Traction	Total
All Traction Wheels	1	4	5
Mixture	3	3	6
All Omniwheel	4	1	5

Reconsidering the type of mogo mech used.

	Grip	Actuation amount	Build ease	Handling	Activation speed	Mogo clamping ability	Weight	Total
Piston	5	3	4	5	4	4	4	29

Motor	3	4	3	4	3	3	3	23
Plough	1	5	5	1	5	0	5	22

Reconsidering our second stage intake

	Consistency	Speed	Weight	Wall Stakes Ability	Total
Hood Mech	4	3	3	3	13
Hook Mech	4	4	4	5	17

Final Decision

Our current plans in terms of rebuilding is to:

- Keep the drivebase the same
- Keep mogo mech mechanism but change lever class from 1 to 3 and tune clamping angle
- Keep first stage intake but lower profile
- Redo ring mech angle
- Add a wall stakes mech (time permitting)
- Get colour sorting working (time permitting)

Intent with the bot

With this version of the bot we hope to improve on the last design, which was fundamentally good but had flawed execution. Furthermore, the design was limited in the possible improvement and additions that could be made (e.g. no room for a ring mech). Therefore we need a robot with:

- Fast and reliable drive train
- Consistent ring scoring
- Strong mogo mech
- Smooth and consistent first stage intake
- Doinker that doesn't touch the bottom ring
- The possibility of adding a wall stakes mech (not definite due to time constraints)

Design

Drivetrain

On the whole, the drivetrain was good, but we do not enough room in the front of the drive train to fully accomodate an optimal first stage intake, due to how the drive was braced. However, there was extra room in the back near the mogo mech. So we decided it was best to simply rotate our drivetrain 180 degrees - so the back was now the front.

First Stage

For the first stage intake we went with simple flex wheel intake rollers. However, in order to improve upon our last design, we changed the intake ramp setting it further into the robot along with having a more favourable contact angle with the ring (the previous bot had the intake ramp hitting the ring before the rollers).

Second Stage

For the second stage, we decided on a central pillar with hooks similar to the first design only with a shallower gradient and a longer hooks stage. At the top there is a much simpler ring guard. This is incase we have time for a wallstakes mech.

Mogo Mech

The JWS mogo mech was good but it requires the room we reallocated to the front of the robot in order to work. Therefore in order to save space, we switched to a class 2 lever, which is a bit less secure but far more space efficient.

Doinker

We changed the doinker structure from a piece of 1x1 to a piece of 2x C-Channel. We did this because the original doinker was not rigid enough and we had to unbend it after every match. The 2x is much stronger due its shape.

? Wall Stakes Mech for our V2 Bot 2024/10/07

At JWS some of the top teams had wall stakes mechanism, which gave them extra leeway in matches and could act as a handy tie breaker in situations where they haven't secured AB. We believe that ignoring this aspect of design for the next contest will be detrimental to our performance. Our criteria for this system are that they must:

- **Reliably** score rings onto wall stakes
- Have a faster cycle time to allow placement of several rings in quick succession

Brainstorming Wall Stake Mechs 2024/10/08

Passive Redirect

One potential solution is to have a trapdoor mechanism mounted on the hook, which is tensioned to allow rings to be intook upwards, but when the intake is reversed it slides backwards down onto a plastic sheet and transitions into a claw or bucket mechanism, which can be used to score rings.
Here is an example of such a passive redirect by 3131V *@DrumrollPlease*

Pros and cons for a passive redirect:

Pros	Cons
• No PSI or motors required • Fairly fast	• Can't intake whilst redirecting

Piston Redirect

Another option we have is a redirect solution, which uses a piston for sorting rather than reversing the intake, before transitioning to a scoring mechanism. The main advantage of this is that it doesn't require the intake to be stopped whilst trying to score rings on a wall stake.

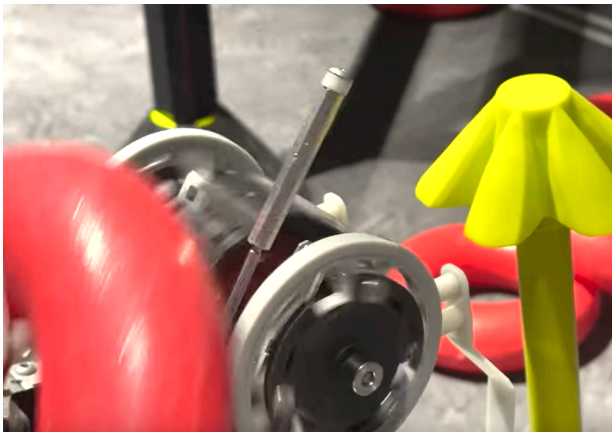
8889A have a solid example of what a piston mech could be. *@vex8889A*

Pros and cons for a piston redirect:

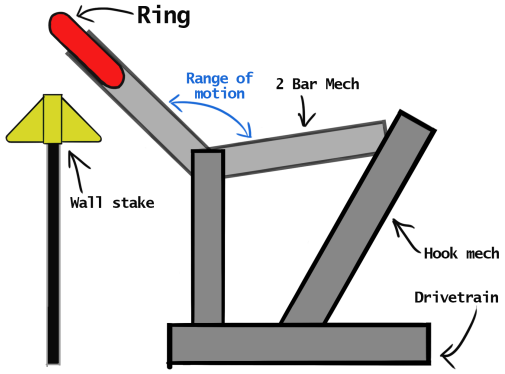
Pros	Cons
• Can intake while redirecting • Pistons make redirecting quick	• Limited actuations • Requires hood based intake

Lady Brown

One of the more recent wall stakes design intially pioneered by *18522R* utilises a two bar mechanism with a pivot at the base and a scoring mechanism at the top. Whilst a ring is on the hook mech the scoring mech clamps around it and it pivots directly off the hooks to score onto wall stakes. The placement of the two bar helps retain a low centre of mass and allows for clean and consistent scoring. This mech works well with a piston based ring sorter positioned at the apex of the hook mech as it doesn't clog up the intake and render it briefly unusable whilst a ring is being rejected unlike other sorter designs.



18522R Piston based ring sorter: When the wrong colour is detected a piston pops up, which alters the trajectory of the ring causing it to not be scored.



Example of what that mech could look like

Pros and cons for lady brown:

Pros	Cons
• Very fast • No redirect required • Reliable ring sorting potential	• Extra motor required

Fish Mech

A very recent mech has enterred the meta, that is very similar to a lady brown mech in terms of design. It also uses a two bar mechanism but the pivot is positioned so that it picks the rings up during their ascent instead of at the apex. Fish mech typically uses a claw to grapple the rings. However, our analysis has led us to conclude that it is much easier to defend against because it's quite slow and the angle has to be perfect.

Pros and cons for fish:

Brainstorming Wall Stake Mechs

2024/10/08

Pros	Cons
• No redirect required	• Eclipsed by lady brown

Designed by: Daniel da Silva
Witnessed by: Daniel Dew

Deciding on a Wall Stakes Mech 2024/10/17

The most consistent wins

Overall we want the mechanism that has the most reliable scoring ability in a variety of circumstances but also we want a mechanism that is fast overall. However, other factors may drastically affect the viability of a solution. We will compare each of these qualities:

- Complexity
- Speed
- Actuation potential
- Intake inteference

	Complexity	Speed	Actuation potential	Intake inteference	Total
Passive Redirect	2	3	4	1	10
Piston Redirect	0	4	3	4	11
Lady Brown	4	5	5	5	19
Fish	5	3	5	4	17

Final decision

We decided to go with a Lady Brown(LB) style mechanism, because we believe that it is the most consistent solution across the board for our robot. Unlike redirects, there is much less need to tune to get it functional, which is valuable time saved. In addition, LB doesn't clog up the hook and stop you from intaking whilst the ring redirects to a scoring mech. A piston redirect, doesn't suffer from that issue but it has limited actuations, which may become a

problem in match scenarios and secondly isn't compatable with a hook based ring mech, which would require us to **completely scrap** that entire mechanism and tune for that instead, which is something we as a team given the viability of other solutions. Whilst, fish mech is probably the simplest to implement, our analysis from watching matches, which we have watched has shown that it is many ways inferior to LB since it is much slower, requires more precision from the driver and also much easier to defend against. Furthermore, LB pairs best with the piston based ring sorting tech pioneered by 18522R, which we are also looking to implement since it is much more consistent than velcoity control base solutions.

Recap of building constraints

Whilst we planed for the likley posibility of having wallstakes mech capabilityes we prioritised having a robot that was viable even without and add it if there was time to add it. thphore there where certain spce constraint aswell as the fact that not every posibility had been though through competly. For exaple there was not enough space to tuck the wall steaks mech as low as is tipical and optimal howerver the was minimal copared to other problems we ran into later.

Building

Gear Ration

We began by instaling gears in a 12 to 72 gear ratio in order to acheave a 6x gear down from 200rpm to 33.3rpm 3.s.f. This gave us prisice and managble controlle we did this because we valued the ability to ensure you score the ring rather that doing faster but less acuretly and potential fail to score it.

Main structure and bracing

For the main structure we dicided to 2 main 2x C -chan arms with grip pads in flaired 3x - chan. The 2x C -chan is extreamly strong and unlilkey to bend. The 3x c -chan is the corect widthe to acomodat and grip a ring well. Flairing the edges increases the tolerace posible with how presisce the ring aproch and hieght has to be. Additionaly we used standoff X bracing to renforec the whole thing. This would help keep the “arms” level with one another and prevent gear skipping on one side or the other witch would cause the wallstakes mech to lose the ability to grip rings.

The problem

During testing (see page 146) we discoverd the due to the leght of the arms and where they began the where incapable of reaching the top of the wall steaks at first this was quite a daunting problem as we could not simply extend the lenght of the arms as they would no longer be in the corect range to acsept rings and moving the end point of the secound stage intake would have been to great a task. We quicly came up with a relativly simple solution to the problem by moving the starting postition of the wallstakes mech further foward on the robot. We did this by slighly extending c-chan it was mounted on and then reataching it.

Introduction

After competing in the Coventry Regional, we have as a team distilled our thoughts and takeaways into a piece of analysis.

The Competition

Qualification

This competitiom didn't have the greatest start, since we lost our practise match against 61150W Andromeda and 1408V Venom, whilst partnered with 30110D. We lost that match since 61150W managed to steal our full stack of rings from us, whilst we were fighting for the positive corner. They placed it in the negative corner, which caused us to lose the match. However, losing a practice match and learning from it is a much more desirable outcome than losing a qualifier or elimination match. We ended up winning our first qualifier (Q5) with 46037A Titan against 1408V Venom and 69922G NUAST Tiger, because [insert strategy here]. Unfortunately, we lost Q12 against 15650C Memento Mori and 46846P Subject to Change due to [insert why here]. From this we can take away that… . From there we won all of our remaining qualifiers and we ended up ranking 6th place at the end of qualifications.

Aliance Selection and Elimination

For Alliance Selection, 13765T MTS_Bubblegum were our alliance partner, with us being 5th seed.

For the R164, we won the match against 6023R StoweBots - Lion and 20785R Rogue quite comfortably, scoring 29-13 in total [since…] . Our next match was the QF, where we played against 6023F Falcon and 20785X Nova. We won against them due to a disqualification involving a violation. In this competitiom, there were multiple surprises in this competitiom, namely 1875N Noodle winning against 15650C Memento Mori in Q39 securring them the 1 spot at the end of qualifications and then subsequently in SF1, where 1875N and 15650C lost against 61150W Andromeda and 66618A ATLAS due to [citation needed]. This was significant for us as we were to play whoever won that in the semifinal match. This turn of events meant that we had to play our SF game very differently. Our match was a very low scoring game with a 1 point difference between winner and loser. Andromeda played negative corners very aggressively [verification needed], an unconventional strategy. During the match we had several technical failures, which caused several mistakes in the course of the game. In the last second of the match we were unable to score a ring onto the wallstakes, which would have been a point swing large enough to win us the match. There were several mechanical and strategic reasons that culminated in us losing that match, given that this bot was largely constructed within the timeframe of around a week and a half, it becomes understandable as to why this occurred.

Skills

Similarly to our last competitiom, we didn't have the time to program any autonomous skills runs and so we settled for doing 1 drivers skill run, in which we scored 29 points, ranking 9th

out of 24 participating teams. However, a lack of an autonomous skils score is a serious concern as it means we don't meet the criteria for excellence.

Interview

We had two judges interviews, our first being a general interview and our second one being focussed about creative problem solving and strategies used in our robots design.

Notes From Team

It is sometimes very useful to note down what each person initially thinks, as often the 'face value' notes are the most useful to keep in mind.

Daniel Dew

Coventry was our best look at the national standard so far, the competition is in fact larger than UK Nationals, with the best teams and steepest competition showing up. This competition was an amazing insight to the required strategies, mechanisms, and even code to compete at the highest level. We can take many tips from this competition, in particular in driving strategy; for example, leaving mobile goals unattended is a must *not*, goals must either be carried or flipped to avoid the points being negated.

Jonah

As a builder I felt that this competition showcased our position within the national rankings and our performance in quals displayed that, with one loss to arguably the best team in the UK I feel like we have done ourselves proud especially given the time constraints we had around the competition and a lack of any real reliable auton. The loss in Semis also showed us a key concept for defence: as we got defeated by an arguably worse robot. On the whole a positive competion however.

Daniel da Silva

On the whole I believe that this was a really important competition for us and our relative successes given our restraints shows that we are a competitive team. Looking forward, we should try be more organised with our time to ensure that we have more time to get the robot working to a standard we are truly proud of.

Thomas

Coventry, being the second largest competition in the Uk, was a privelege to attend and a tremendous learning experience. Managing to get to the semifinals again was a huge success for us, especially since we were competing againts some of the highest level teams in the uk. Although it was dissapointing to lose the semifinals, it was an extremely close match with a closely called negative goal at the end which ulitmately caused us to lose. Despite this, placing 6th with a 6-1 Win-Loss ration wasa tremendous for us, and we learnt a lot from both a strategical, and build aspect of our robot. We are looking forward to implementing some new changes adn hopefully getting qualified with our next competition.

Aubert

I think that Coventry was very instructive. It was great for an experience gaining point of view. We competed on an essentialy nationals level. All the best UK teams were present and we were able to compare ourselves to them. What I in particularly took from it was that our robot was overall viable but our lack of tested autons held us back from achieving as highly as I believe we could have. In future, I plan to keep in mind autonomous compatibility when building the robot.

Results and Reflection

Overall for our second competition and the context of the timeframe we had prior to this competition, we did exceptionally well. Making 5th seed in arguably the UK's largest tournament, which for reference we made 14th seed at this competition last year, ironically also with 13675T as our alliance captain. Additionally, we managed to win the Create Award and did all of this in under two weeks.
Moving onwards, we need to ensure that we have **enough time** to fit all of the changes, which we plan to make.

Analysis of Our Semi-Finals Match

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae sine metu degendae praesidia firmissima. – Filium morte multavit. – Si sine causa, nollem me ab eo delectari, quod ista Platonis, Aristoteli, Theophrasti orationis ornamenta neglexerit. Nam illud quidem physici, credere aliquid esse minimum, quod profecto numquam putavisset, si a Polyaeno, familiari suo, geometrica discere maluisset quam illum etiam ipsum.

? What needs Improving 2024/11/18

Optimisation

The ultimate plan for this rebuild is: Optimisation. We want to improve on and redesign the weakest aspects of our robot's design and additionally we want to take the parts of our bot, which worked well to the greatest potential. Since our current bot was built largely in the span of a week there are many things, which were rushed or cut for the sake of time. The main issues of our current design were:

- The mogo struggled with clamping and aligning
- High centre of mass
- Too heavy
- Intake would jam occasionally
- We didn't have enough time for autons

The most consistent/best part of our bot was:

- The wall stake mech
- Ring mech scoring
- Daniel's driving

Moving onwards

Our next competition is the 4th January, in which neither Jonah nor Aubert can make due to holidays, and we have approximately 1 month and 3 weeks until then. In this time, we need to relax as we are all very tired from this competition and 4 out of the 5 members on this team are involved in the school musical, which takes up almost the entirety of the week commencing the 18th November.

Once that is done, we can discuss what problems require the most urgent attention and what strengths can be improved the easiest. After we have this discussion, we can move into the design phase, creating mockups and eventually creating CAD for what we want our new bot to become.

Once we have CAD, we can work on dismantling and reconstructing our robot. We have decided in advance to keep our current drive train the same since as mentioned previously we spent a lot of time getting that right. However, we will spend some time checking the friction of our drive and the power that it draws.

With our brand new robot, we can work on driver practice and autonomous routine's in parallel. As soon as our autonomous routines are consistent, we can focus all of our attention on drills/practice and maybe even a scrimage with 1875N Noodle with hopefully a week or so to spare before Birmingham.

Brainstorming Improvements 2024/11/20

Fixes

This first section is focused on improving things, which didn't work particularly well in our current bot.

Improving the mogo

Our post JWS bot had issues with the mogo clamping and aligning. Between JWS and Coventry we switched from utilising a Class 1 Lever to using a Class 3 to clamp because of spacing issues during the CAD/rebuild phase. However, in terms of match performance, we sometimes struggled clamping mogos, which is detrimental in a match. Therefore we are reconsidering as to whether our redesign should use a class 1 or class 3 lever.

First Class Lever

Pros	Cons
• Can offer mechanical advantage • Effort can go inside, which can be more space efficient	• Slower • Might be tricky to fit in - if limited space inside the bot

Third Class Lever

Pros	Cons
• Gives speed advantage	• Provides less clamping force • Since Load and Effort are on one side this can increase complexity and footprint

> **Note**
>
> Ultimately, there's little difference in terms of performance between a class 1 and a class 3. It is more situation dependant, where the overall design within the context is important, rather than any subsystems capabilities on paper.

Improving stability/weight issues

There are several methods which can be used to improve our robot's centre of mass. Firstly, removing weight off the top and shifting components to be as low to the ground as possible greatly helps. Minimising overhangs and widening drive base will also help with stability. Finally, we can additional weight to the bottom of our drive, as that will shift centre of mass downwards. We did this last year and we found that the the best way to do this in OU was to box C-Channel and put nylocks wrapped in nonslip mat held down by zipties, in order to shift our centre of mass forward.

Removing/Shifting existing weight down

Pros	Cons

- Greatly helps improve stability
- Decreasing mass can help with motor performance (Power = mass * acceleration * velocity)
- Less weight reduces coefficient of friction

- Adds constraints to consider during CAD, build and test phases
- Loss of weight means bot is easier to push around

Minimising overhangs/Widen base

Pros

- Wider base will make us harder to tip from certain directions
- Fewer overhangs make us overall more stable

Cons

- Widening drive base would require redoing a lot of work
- Wider drives have their own caveats
- Few overhangs to start with

Adding additional weight

Pros

- More mass means more momentum
- Can quite substantially shift down our centre of mass
- More weight due to mass will cause us to be harder to push

Cons

- Increased weight will cause greater friction coefficient
- Mass is defined as resistance to acceleration (Newton's Second Law)
- More mass will harm motor performance (Power = mass * acceleration * velocity)

Ways to fix intake

The main problem with the intake was that it would jam occasionally and struggle to suck in rings. There are several approaches, which can help curb this. Firstly, experimenting with different flex wheel configurations can help increase traction with more flex wheels tending towards more grip. In addition, we can try redesigning intake chain so its on the same axis as the chain. In addition, we can experiment with the use of plastic to make our intake lighter yet still durable. This weight saving doesn't necessarily help with the inherent problem with the intake, but weight saving is generally a good idea.

Improvements

This section is dedicated to considerations to improve and upgrade existing systems, which already well.

Wall stakes mech

Our current wall stakes mech is based off the design by 18522R, which was arguably the most consistent part of our robot and additionally it scored **quite quickly** which is important, as whilst playing wall stakes your bot is very susceptible to being pushed - scoring faster mitigates the chance of this. However, we have seen an interesting new design, which came

out early this month, from *1028A* . There wallstakes mech or direct mech as they call - utilises a valve or 1 way lock mechanism on a ring bucket that is mounted on their hook mechanism. What this allows is for rings to pass cleanly from the intake to a mogo without interferance or it can allow them to score onto wallstakes without redirecting first, which would waste time. [Drawn diagram of direct mech]

Here's the pros and cons for remaining with a 18522R based design.

Pros	Cons
• The golden standard • Known to work	• It is quite heavy

Pros and cons for moving onto a 1028A inspired design.

Pros	Cons
• More compact • Easier to score with • Can be faster	• Similar in performance to 18522R's design

Ring scoring

A nice adjustment to our bot would be being able to score onto aliance stakes using our hook mech. This requires redoing the angle of our hooks for the CAD/build phases.

Deciding on fixes

Deciding on the mogo

For this design change we are considering:

- Clamp ability
- Build complexity
- Ease of implementation

	Clamp ability	Build complexity	Ease of implementation	Total
1st Class	4	3	3	10
3rd Class	3	4	4	11

Weight and stability

For this redesign, we are considering:

- stability net increase
- viability

and additionally we will consider any side effects each of these solutions provide.

	Affect on stability	Viability	Total
Shifting/Removing Weight	3	4	7
Widening base/Overhang reduction	2	1	3

	3	4	7
Adding weight			

Removing excess weight will increase our motor performance and reduce friction coefficient but will reduce our momentum and mean we are easier to push around. Adding weight on the bottom, has the inverse effect to this but overdoing this could make us unviably heavy. Shifting weight down requires conscious effort during CAD and build phase. Widening our drive base would be impractical as it means we have to completely restart our drive and redo the friction on our drive, which is time we do not wish to spend. In terms of overhangs, our bot doesn't have any permanent overhangs and the temporary ones such as the wall stakes mech or the doinker are insignificant in terms of % mass.

Deciding on intake improvements

We have decided that we are going to try as many things possible to improve the intake - in terms of its core problem but also just general upgrades. This consists of the changes proposed in the brainstorm section: different flex wheel configurations, chain position and use of chain reverser, plus a focus on weight saving. However, much of the improvements will come from tuning and tinkering once fully built.

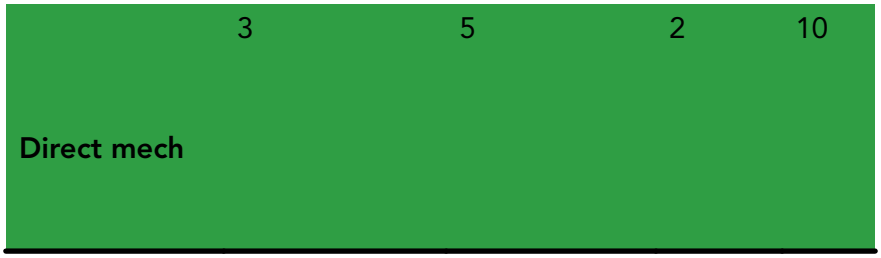
Deciding on improvements

Two bar or direct mech

Here's a relative comparison of both a two bar mech and a direct mech, based on the following criteria:

- speed
- compactness
- weight

	Scoring speed	Compactness	Weight	Total
	3	4	2	9
Two bar mech				



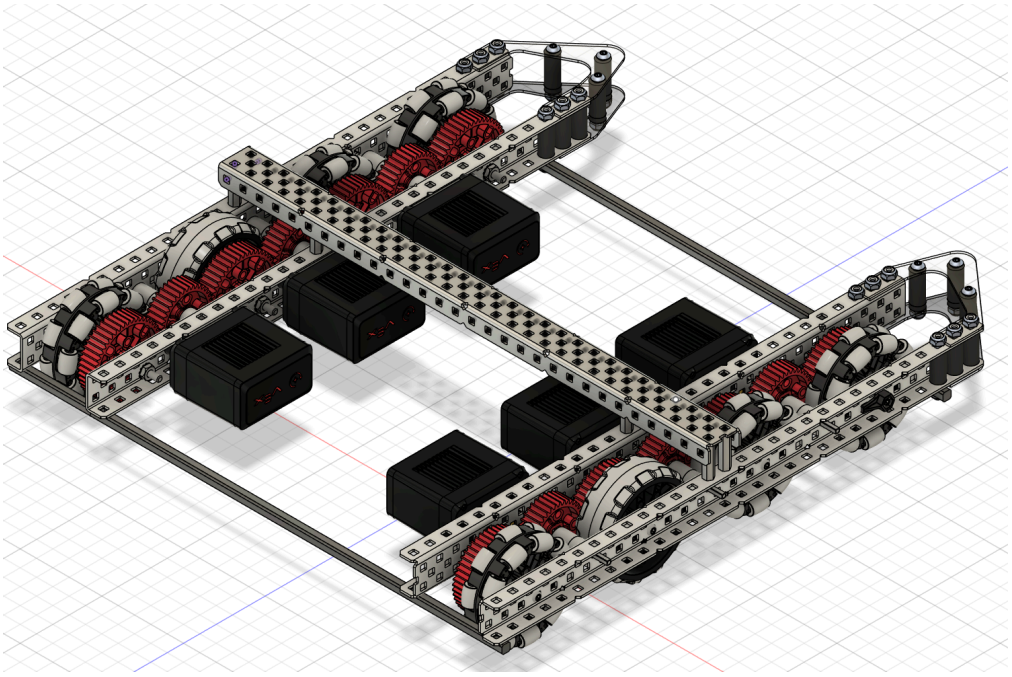
Final Decision

In conclusion, our proposed redesign will focus on improving the mogo clamp, improve stability by shifting down weight and trimming off excess weight and will put a special focus on improving the intake via tuning. In terms of wall stakes mechs between a direct mech and the already established two bar mech, as a team we believe that the direct mech has a lot of potential and since we have enough time to work with it and fully figure it out. An additional bonus is that is a lot more compact than existing two bars. However, we know that two bar mechs are a consistent and safe fallback.

Starting the New Design

Recycling

When beginning a new design cycle, there is often many transferrable aspects of the last design. In this case, we can reuse a large amount of the existing drivetrain, as the width and the gear ratio are remaining the same.



The CAD of the recycled drivetrain from both previous design cycles.

Modifying Drivetrain Design

Shortening drivetrain

While shortening the drivebase does not directly benefit drivetrain performance, it allows us to further shrink our footprint, this can be very useful for many reasons:

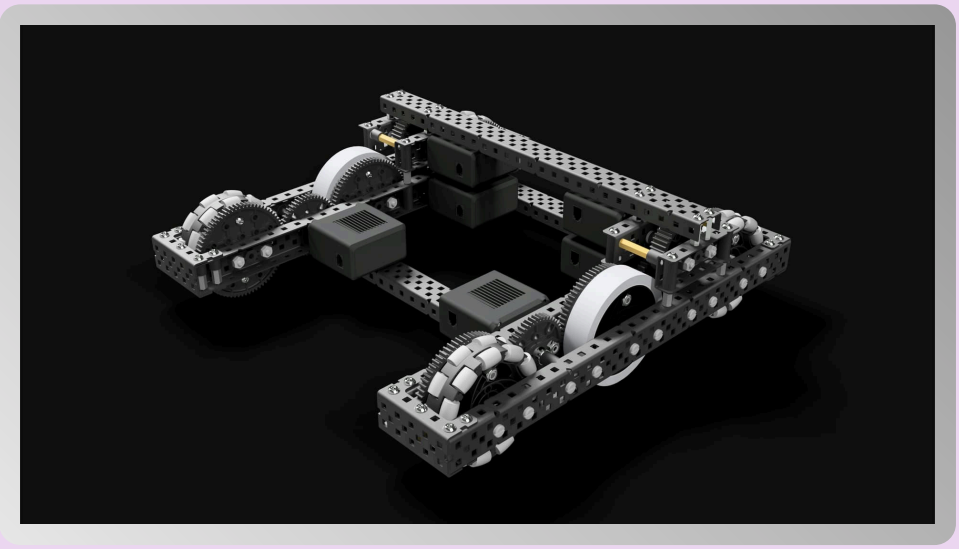
- Field Manoeuvrability: being smaller (even if its marginal) can greatly increase our ability to navigate through and around field obstacles – as well as outmanoeuvre opponents.
- Expansion: Having a smaller footprint within the allowed 18x18" size limit allows our expansion (in desired direction) to be greater.
  - Example: robot with starting size 18x18" allows for 6" in one direction. Whereas, a robot that has starting size of 18x15" allows for expansion of 9" (assuming in direction of smaller dimension)

Stacking motors

One of the design choices we have not yet considered is whether to employ stacked motors to aid with weight distribution and space. There are typically 2 ways motors can be stacked, full stack and partial stacks: partial stacks are where one motor is moved from the base and

elevated above the middle (or back/front) motor, this allows for more space to be freed in the back or front while not sacrificing too much build complexity; a full stack of motors is where all 3 motors – on each side – are stacked in one column, this is considerably more complex and prone to friction but can drastically free space. Full stacks can also be easier to PTO[1], as accessible gears would be closer to power hungry mechs; this *could* be useful as we look into potential climbing mechanisms later.

Example



An example of a partial stack, where the back 2 motors are stacked in a column [2]

Above, there is an example of a partial stack of motors, where the back 2 motors are put in a column, in this case, it allows for significantly more space in the front.
A full stack would involve taking the front 2 motors and elevating them onto the top of the existing stack.

In our case, this can be used to allow for space for various mechanisms, or just to act as empty space to, for example, get close to the corner post of ladder (top potentially climb)

Pros and cons of stacked motors

Pros	Cons
• More space in front or back; something that may be strictly necessary	• Higher COM • Added build complexity ▸ Potentially more friction

[1]Power Take Off (*PTO*) mechanism, used to transfer power away from motors.

Final Decision

Ultimately, for this design cycle, we have decided to employ a partial stack of the motors. This will allow us to drastically increase space to facilitate our possible future ladder climb plans. While this decision may not directly help the performance of the drivetrain, we believe it is a necessary step to future-proof our robot.

CADing proposed modifications

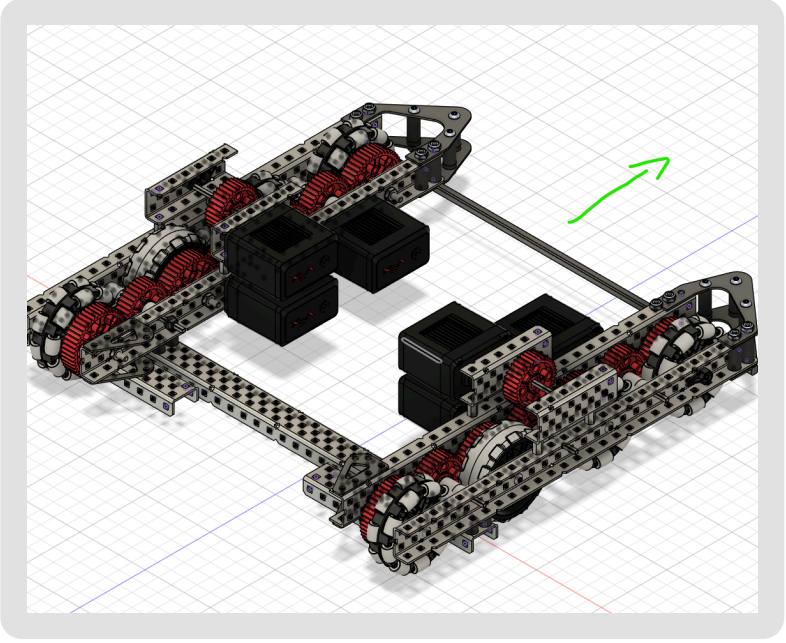
So far, the modifications are only proposed, the point of CAD is to prove such proposals are feasable and to then later provide a basis on which to build on.

Shortening drivetrain

This step is incredibly simple, it simply requires replacing the 30-long drivetrain pieces with 27-long pieces. We decided on this length becasue it was the maximum we could shave off without jeopardising structural strength.

Stacked Motors

This step is also simple, as it only entails creating a raised platform on small standoffs – where the motors can be moved onto a stack.



The drivetrain CAD after implementing proposed design changes – front denoted by green arrow.

Note

Notes on CAD

- The shortened drivetrain is not particularly evident, but the frame has been shortened by 3 holes (1.5”) and the funnels have been moved back one hole (0.5”).
- More evidently, the motor that would be at the back is moved onto a stack with the middle motor; this allows us to ‘envelop’ the corner post of the ladder, potentially facilitating our future ladder climb plans.

Designining Complex Moving Mechanisms

Motor Distribution

With this design there are several complex design strategies we can employ.
For example, there are a few ways to organise the distrubution of the remaining 22W of motor power; we could, for example, allow 1x 11W motor for our intake mechanism, similar to V2; therefore allowing 1x 11W for the direct high stake mechanism.
If we wanted to dedicate more power to the more complex chain bar required for direct mech, we could use only 5.5W for the intake/hooks, and therefore use the remaining 16.5W for the chain bar.

Motor Sharing

We have decided that both the intake/hooks and the chain bar neeed more than 11W; this is normally unobtainable using normal motor distrubution, however we wish to emply the method of **motor sharing**.

Example

What Is Motor Sharing

- Motor sharing is the technique of using the same motors over the different mechanisms.
- Obviously, we dont wan't the intake and chain bar lift running at the same time, so we can employ a **ratchet** mechanism.
  - Rachet mechanisms allow for a system to be mechanically disconnected while the motors run one way, then engage when the motors are reversed
  - In this case, we can allow the intake to run at all times, but when it is reversed; the ratchet engages and the chain bar lifts.

An example of a ratchet mechanism (left) [25].
Here, the metal gear spins no matter what – in one direction, the screw freely skips over the gear (therefore the green sprocket remains stationary[2]). In the other direction, the screw catches, and rotational torque is transferred to the large gear

[2]The gear has a round insert and is therefore unaffected by shaft spinning.

? Big Dilemma 2024/12/02

The Problem

We have come into school today and we have been told that, the robotics space needs to be shut and that we can no longer use it, as it being used as an exam hall for the mocks going on in our school, after an initial promise that we were guranteed this space. In addition, only two out of the five members are able to go to Birmingham due to holidays and other prior commitments.

What this means for us

Today is the Monday 2nd of December, school ends in two weeks on the 13th. This means that we are going to have to spend approximately a week out of dwindling time till Birmingham on the 4th, in order to pack up, find a temporary relocation and resume work on the building of the robot. Ultimately, this delay will put us behind schedule by at least two weeks and effectively we have less than a month to get everything done. This means that we can't achieve everything that we wanted to as set out in decide. Three things are clear:

- We will not have time to tune a direct wall stake mech
- We will have to work through Christmas break
- We will not be able to tune autonomous due to a lack of a field.

Overcoming this Problem 2024/12/02

How we overcome this

Before continuing we have to reign back, the scope of our bot and fix/do **only what is critical** within our timeframe. In order to overcome this challenge, we are going to try do as much work on the bot, whilst keeping on top of school, before term ends. Afterwards, someone will take the robot home for Christmas and work on it when they can, and those who can will meet up in orde to help. After much discussion, we have worked out what needs to change. Since we no longer have the time to comfortably get it working we can no longer utilise a direct mech for wall stakes. Here's a new list of proposed changes below:

- Improved mogo clamp
- Redesigned Intake
- Adjusted Centre of Mass

> **Note**
>
> Unfortunately, we were unable to go to the Birmingham regional since 3 out of our 5 team members were unable to make the competition due to prior commitments. Therefore, we decided that it wouldn't be worth it as we wouldn't be able to have a full drive team. Instead we felt it was worth concentrating on getting our V3 bot working to high standards.

Recap of all issues with curent robot:

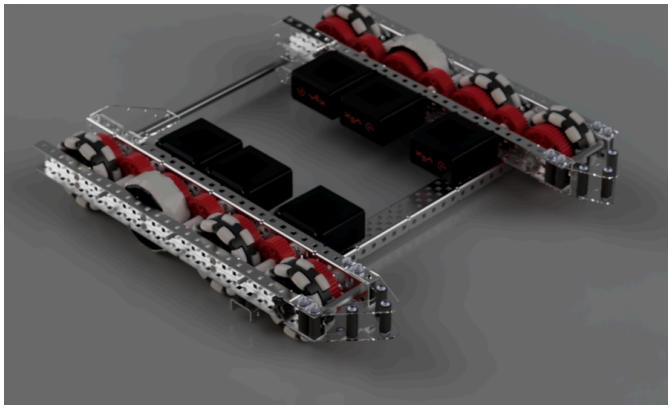
- Center of mass is too high (tips a bit under rapid acceleration)
- Robot is too heavy
- Intake ramp was bent manually therefore the curve is suboptimal
- Mogo mech was weak

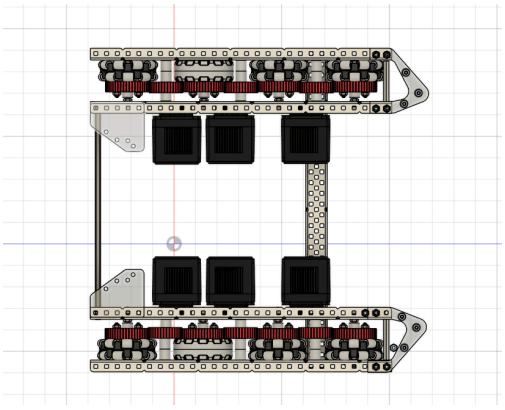
Design

Drivetrain

We decided to keep the base drivetrain because we are very comfortable with the speed and acceleration we get from our current gear ratios. The drive's size is 30x30 holes plus the length of the intake funnels. We want to cut down the robot size slightly since a smaller bot is more manoueverable and it gives us more room to expand in our chosen direction. We are going to cut the excess 2 holes of metal we have on the front of the robot. This will reduce the space available for the intake but not reduce it enough to be a problem.

To better accomodate the first stage intake, we have opted to get rid of the front brace. After carfully weighing the pros and cons, we decided that brace was largely redundant due to the 2 other braces on the bot. Additionally it reduces the mass of the robot, which leads to better acceleration given that acceleration / ms^{-2} = $\frac{\text{resultant force / N}}{\text{mass / kg}}$. Since our drivetrain's pushing force is limited by the motor's power output, the lighter our bot is the better its acceleration will be. We also plan to use 2 odometry pods for accurate horizontal and vertical positioning, which allow our autons to be more consistent.





First Stage Intake and Transition

Ramp:

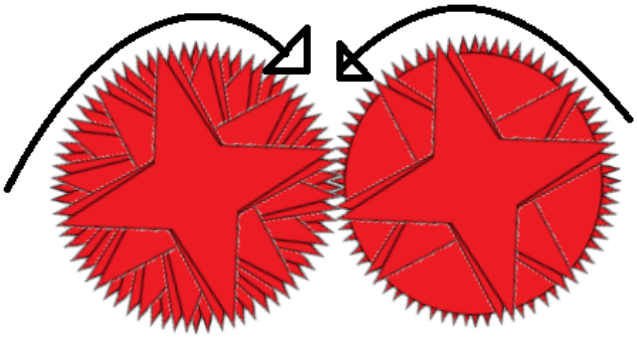
For the ramp we decided to go with pieces of laser cut Delrin rather than the heat bent polycarbonate plate, which we used last time. The greatest advantage of this method is that laser cutting is more precise, which means that ring alignment is more consistent. Additionally, the curvature of the laser cut pieces matches the ring better making for a smoother and more efficient intake process.

First Stage and Intake Rollers

We made the first stage on a level plane as this gave more consistent traction while intaking the rings. We also used Delrin to brace the whole thing.

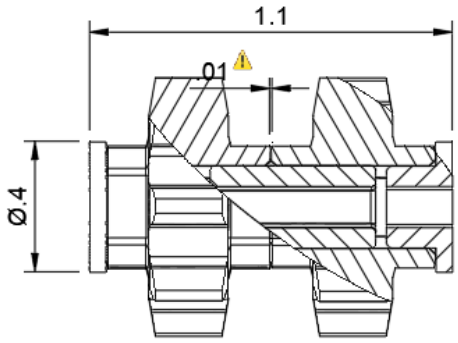
Gear Reversal

We elected to have both the first and second stage intake be powered by the same motor, which means that we can allocate one motor to the wall stakes mech. The only problem with this approach is that the intake rollers and hooks have to spin in opposite directions. We also faced this problem in V2 of the robot. A geared chain reversal was a simple and effective way we used to solve this issue.



Gear Reversal

A chain reversal presents 2 main problems: friction and tensioning. Typically, when dealing with the meshing of 6 tooth sprockets and gears you connect them using low strength shaft, which are very bad for friction. We solved this problem by using a creative combination of inserts to couple a gear and a sprocket together as shown below:



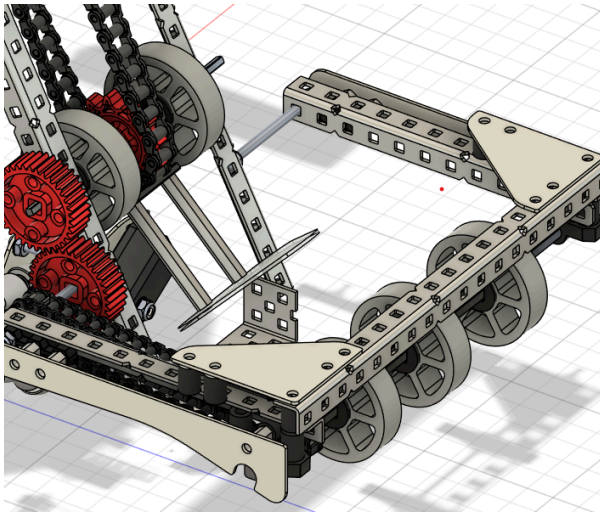
Gear Reversal

This allows us to use screw joints, which produce far less friction than the low strength shaft joint. The chain needs to be under a bit of tension in order to work properly otherwise the

chain will simply skip. This normally isn't a problem although there are times that the intake needs to be able to lift up in order to grab the top ring on a 2 high stack during autons. Therefore the chain needs to be under tension while in 2 positions.

Most people solve this problem by using a tensioner: a piece of metal on a screwjoint with spacers to reduce friction. It is rubber banded outwards, to provide tension and remove slack. Bicycles have a similar mechanism used to keep the chain under tension no matter which gear that the bike is in. We chose a different solution as tensioners can fail which could potentially **ruin our ability to score**. Our solution was to have the chain go to and from the intake roller while spining about the joint about which the first stage intake is mounted this meant that the 2 sprockets remain equidistant regardless of the angle and this design is less likely to fail.

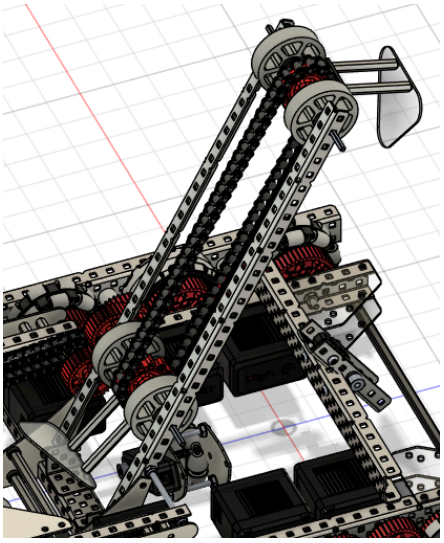
Example of tensioner on bike



Gear Reveverasl and flex roolers

Second Stage Intake

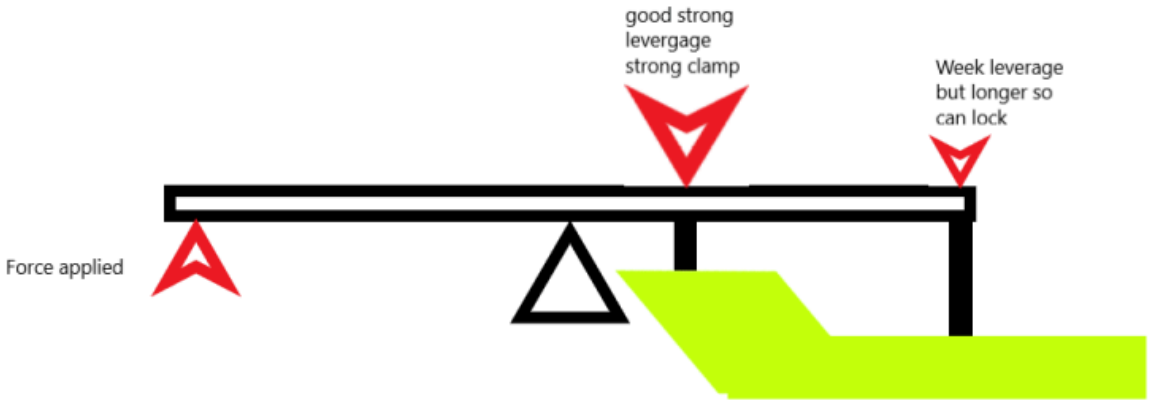
The second stage intake remains largely unchanged, being made of many chain links with hooks to carry the rings to the top of the conveyor. It is run on 24 tooth sprockets, as this gives the preferred chain speed. Futhermore, we put flex wheels at the top of conveyor, this helps score rings onto the mogo since flex wheels provide a lot of traction. The frame is slightly altered in the details but overall remains the same: central pillar now made of 1 x 1 instead of 2x C-Chan because it is not weight bearing therefore 1 x 1 is strong enough for this purpose and much lighter than C-Channel.



secound stage intake

Mogo Mech

For the mogo mech we reverted to a 1st class lever, while the previous lever was more compact, the trade off in required PSI was not worth it, as it drastically limited the number of actuations available to us during a match. It also meant that when accelerating the mogo would tip backwards slightly, which made ring scoring very inconsistent. This mogo clamp was designed with the flaws of the last one in mind. As a result the design has a big emphasis on taking advantage of leverage so it uses very little PSI, whilst still providing a secure grip. We are very proud of our new system, which was designed by our team leader Daniel Dew, who also created the CAD version of it as seen below:

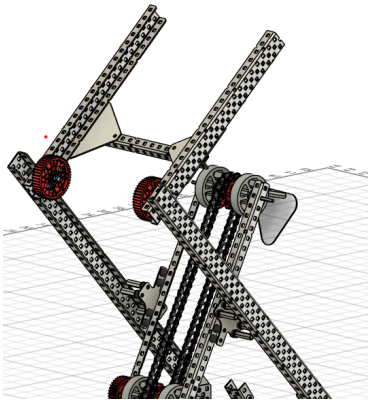


Mogo mech forces

The mogo clamp has 2 sets of prongs: a first set of prongs inset further into the robot - these have very good leverage and so help us to use less PSI. A second set of prongs serve simply to secure the mogo. These prongs apply on pressure on the mogo and instead act more like a seat belt or hard stop and prevent the mogo from misaligning.

Wall Stakes Mech

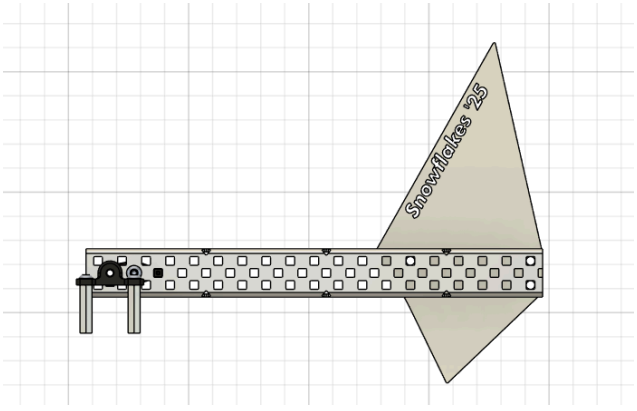
The wall stakes mech was designed with extreme precision as we wanted a wall stakes mech that was able to rotate through its full arc without going out of size limits. This is important to us because it allows us to score on the alliance stakes (very useful for autons) and also allows us to tip mogos over. The reduction in sizing prevents the possibility that we get disqualified because of a hard stop failure.



Lady Brown

Doinker

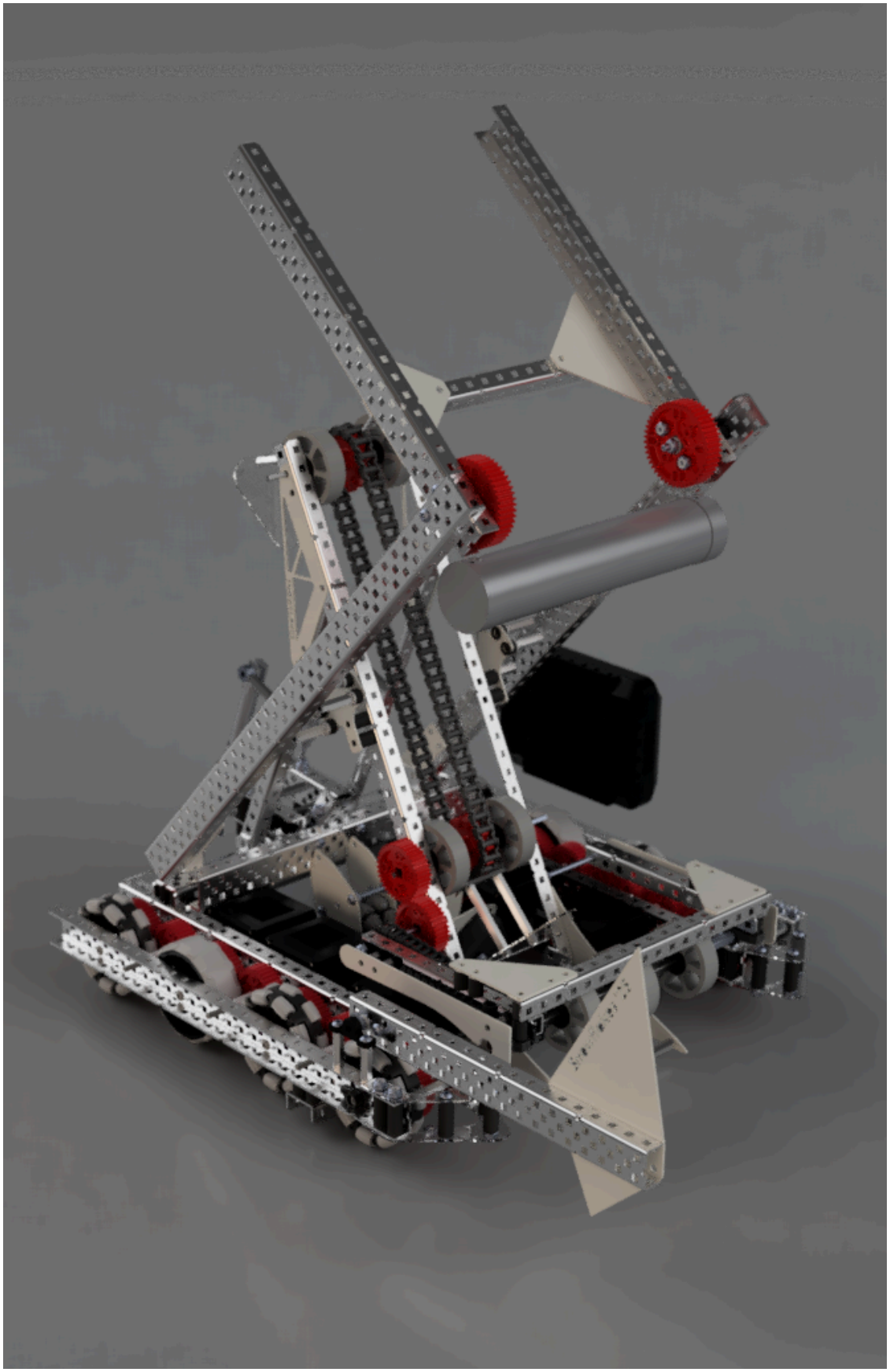
The doinker was unchanged with the exception of geting new plastic able to push all the rings rather than the top 3. As we found that there was a design flaw since it restricted our ability to clear positive corners and place mogos there.



Doinker

Designing our V3 Robot

2024/11/20



Designed by: Aubert Seysses
Witnessed by: Daniel Dew

Building the Bot:

Drivetrain

We rebuilt the same drive, which we where using before from scratch but with more attention to detail and managed to get the friction down to 0.1W, which is essentialy negligible.

First Stage and Chain Reversal

The whole construction of the first stage intake, including chain reversal was flawless thanks to the CAD done in advance. But there was only one small problem, which was with the ramp bracing, where the standoffs were too close to the slope, which slowed and messed with the ring transition. It was not a big issue and easily resolved by drilling a new hole inset into the bot slightly more.

Mogo Mech

The mogo mech was largley flawless because we had CAD done in advance so the mech only required minimal tuning. We only needed to change the second prongs, since they were abrasive enough to sometimes catch on the edge of the mogo causing it to not score rings properly. However, all we had to do to resolve this was sand down the threads on the screws on the side contacting the mogo and the issue was resolved.

Second stage intake

In the second stage intake, when we first built it we encountered several issues: the ring only scored roughly 1 in 8 times. After extensive testing, we concluded that it was due to the placement of the hooks being roughly 0.5in too low. This in of itself was not terible, we just attached a piece of 1 x 1 to extend the length of the central pilar slightly. After quite a bit of tuning and tweaking we had scoring working within tight margins.

Wall Stakes Mech

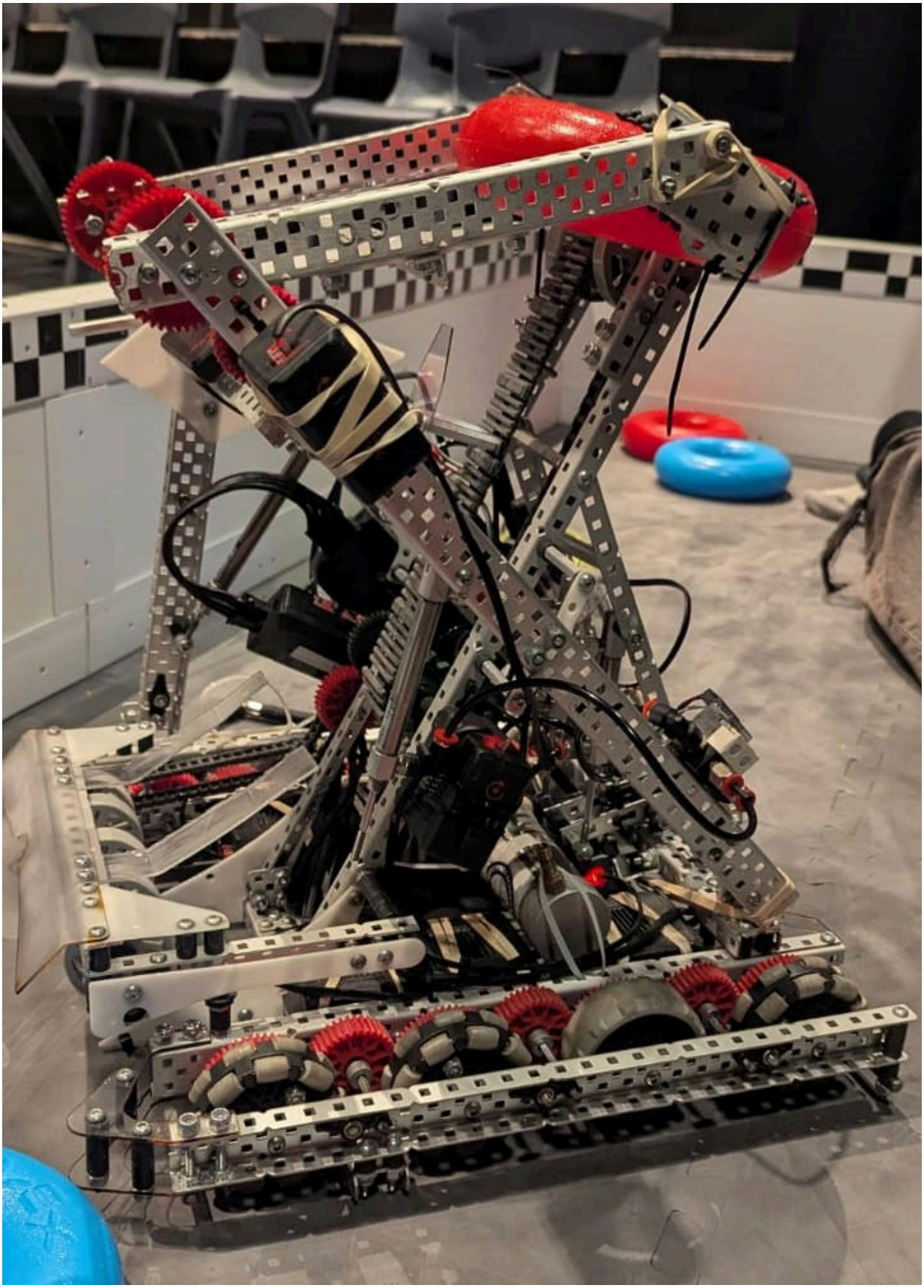
The wall stakes mech was based on the position of the end of the second stage intake, which moved due to a slight error in the CAD. However this had a knock on effect on our CAD, which meant that the wall stakes mech needed to be changed. We resolved this by tuning, clipping and doing other micro ajustments (e.g. carefully cutting the metal in a way so that it would slot in at a diagonal because that is the only angle which can reach the top of the stakes). Tuning this was probably the biggest job apart from adjustments to the drivetrain. But eventually we succeeded in fixing it and could score with it reliably.

Doinker

The doinker was easily executed as it was mostly unchanged from before.

Building our V3 Bot

2024/11/20



Designed by: Aubert Seysses
Witnessed by: Daniel Dew

Introduction

After competing in the Stowe Regional, we have collated our thoughts about how it went.

The Competition

Qualification

We won our practise match, 71611A and us vs 54345A and 46037D but we got an autonomous DQ and we had problems grabbing the mogo at the start of the match with the final score being 13-19, the final score was much closer than we'd like it to be.

Our first qualification match Q3 was 3110C and us vs 14109B and 3116Z. We won our autonomous and our driving was much better getting a full mogo quite quickly, but there was a couple rings which didn't properly score. We ended up going for a second mogo managing to half fill that missing a ring scored but regardless winning Q3 26-0.

Our next qualification Q5 was 6023R and us vs 3110C and 46037D. We won the autonomous for Q5 and played aggressively. We almost completely filled up our mogo in drivers practise and cleared out positive corner managing to get a sneaky 6th ring whilst waiting for corners to be protected. We managed to half fill a second mogo before time wining 30-9.

Q9 was us and 6023R vs 11552A and 54345A. We had an autonomous tie since 6023R had an issue with their motors meaning they didn't move off the line. We used our normal strategy and managed to win 23-8.

Q16 was us and 46037D vs 71611A and 6023J and Q19 was us and 46037B vs 6023S and 14109A.For both matches we played with similar strategy and tactics. We won both 25-5 and 29-3 respectively.

Our final qualification was Q21 with us and 282S vs 3116G and 3110D. This match started well, we managed to get a full mogo relatively quickly. However, 3110D were blocking us from scoring in one postive corner and the other positive corner was being contested by our alliance 282S meaning there was no corner for our stack to go in. Daniel decided tip the goal so it couldn't be grabbed by the opposing alliance. However, 3110D were able to untip our 6 ring goal using their doinker and managed to get it in the negative corner. We tried to get it out using our mogo instead of our doinker, which ended up wedging it in further. Fortunately, 3116G interferred with one of the positive corners, whilst they were protected. This meant they were immediately DQ'd and so we ended up getting the win point. We finished qualifications in 1st place, which means that we are guaranteed 1st seed. At this point we felt that either 282S and 3110D would have been our best alliance partner.

Aliance Selection and Elimination

We allianced with 282S because we felt that they would have worked with our robot better and allowed us to play defensively. We want to play defensively because we don't want any chance of being DQ'd or getting mogos stolen like in Q21. In our QF match, 282S rushed positive corner and defended it, whilst we filled up a mogo to 6 rings.We won 31-0.

Our SF match was played well and we ended up winning 33-0.

Finals 1 was against 3110D and 6023R and we ended up getting an autonomous tie. However, before driver control our bot started moving forwards on its own, whilst not recieving any inputs. It turns out that we had a faulty connection, so they restarted the match. Finals 1(rerun) again resulted in an autonomous tie because our bot didn't score a ring on the auton. 282S spent almost the entire match guarding one of the positive corners, whilst we ended up filling up a mogo entirely and we waited for 15 seconds to go off. At this point, we knew we had won but we decided to score additionally onto wall stakes for some extra points. We won 20-11 and hence won the finals.

Skills

For skills, we did one drivers run, we scored 21 points. Unfortunately, our intake broke as our battery got jammed, which limited how much we could score. We didn't win skills champion but if we had the time prior to this competition, we would have had been able to get an autonomous skills working and we would have almost certainly won skills champion.

Notes From Team

It is sometimes very useful to note down what each person initially thinks, as often the 'face value' notes are the most useful to keep in mind.

Daniel Dew

Jonah

Daniel da Silva

In my opinion, this was an amazing competition and we did phenomenally. The biggest thing that needs doing now before Essex in my opinion is work on our autonomous. I decided to write this reflection whilst at the competition and edit it afterwards. This seems to have worked quite well and saves time so we shall most likely do this in future. However, it was especially necessary to do this given that we are competing in Essex tomorrow.

Thomas

I think strategically, we started very well but got better over the course of the competition. I believe that our final game was where our strategy was best: it was simple but effective and it worked. Overall, I think that this was a good tournament and we played reactively and with good strategy.

Aubert

Results and Reflection

In conclusion, the Stowe Regional was a really good competition for us as we have securred our first tournament championship ever. Getting this award from winning finals and coming

first in qualifiers means that we succeeded in one of our season goals and are still on track for another one. We have succeeded in getting a place at nationals and have still made top 5th Seed for every competition, which we've went to, meaning we are still on track for: A Force to be Reckoned with. From here we plan to get in as much practise as possible in order to be at our most competitive for worlds. However, we plan to be more careful as well as we still had a few close calls - not all of which were our fault.

Introduction

After competing in the Essex Regional, we have decided to summarise the event.

The Competition

Qualification

Our first qualification Q3 was quite interesting, we were with 8346D and against 20785A and 10173A. Our alliance blocked the negative corner, whilst we filled our mogo and got it in the other positive corner. However, our driver left the corner undefended early thinking that it that it was 16 seconds left instead of 26. 20785A managed to descore our mogo into the negative corner. However, Daniel in the last seconds of the match was able to descore their mogo in the other corner and also managed to tip our mogo with the doinker so it was no longer in the corner - we learnt this trick from Q21 in Stowe yesterday.

Our second qualification Q7 was problematic, we managed to get autonomous bonus but our mogo mech wasn't working properly and so we struggled to score rings on the mogo. We ended up getting 4 of our rings plus one of the other rings on our mogo and failed to score a ring on our mogo mech. However, we drew the match because of our autonomous bonus. Q11 we also had consistency issues with the mogo clamp, which is beginning to be a pattern at this point.

Q18 was a good but tight match, our auton scored alliance stake and grabbed a mogo but we weren't aligned to score a ring. The most notable thing was that we tipped over our mogo and stole the oppositions mogo from the negative corner.

Q25 was against 15650C Memento Mori and it was a very close match. They got one one corner whilst we got the other…

Our final match Q27 we won and nothing particularly of note happened.

We finished qualifications in 9th place with 3 wins, 2 loses and 1 tie, which is not as good as we'd hope for.

Aliance Selection and Elimination

We accepted 13765T MTS_Bubblegum's alliance proposition, which meant that we formed the 4th seed. In quarter finals we played against 17711C and (), we won 18-17. However, they double possessed, which would have been match affecting had we lost. In semi finals, we played against No 1 Seed 13765X MTS_R3START and 17711E Entropy. In that match, we played to the best of our ability with 13765T playing superb defense. Near the end of the match there was a close call where R3START touched the positive corner when protected, which could have been a DQ but the ref made the call that it wasn't, despite a lot of arguing from our alliance.

Skills

We enterred one driver's skills and got a score of 22, placing us at 12th out of 14, meaning we didn't win skills.

Notes From Team

It is sometimes very useful to note down what each person initially thinks, as often the 'face value' notes are the most useful to keep in mind.

Daniel Dew

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

Jonah

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

Daniel da Silva

In my opinion, this competition wasn't too bad for us

Thomas

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

Aubert

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

Results and Reflection

Overall, the competition wasn't the best for us but it wasn't outright terrible - despite winning no awards. At the competion, 15650C considered us for an alliance partner since they ranked 7th and we ranked 9th but due to the match schedule, some teams placed much higher than they should have and others placed much lower than they deserved. For many teams this made getting the alliance we wanted much harder than it should have been. Additionally, we managed to improve the consistency of our mogo clamp over the course of the competition, which meant that our ring scoring was much more reliable. In addition, it made good driver's practise for Daniel against some of the best teams in the UK.

Manual Update 3.0

On the 28th of January, we werer blessed with our highly anticipated second major game manual update. After the first one changing the corners to traingels, as well as many other major rule changes such as allowing rings on mobile goals to be scored even if they are touching teh ground. many members of the vex community were very excited to see what this update would have in store for us, and how our game strategies and tactics would change. Needless to say, the game manual did not dissappoint.

How High stakes changed:

It would be an insult to the manual if I didnt start this page with the rewrite of rule SG11; positive corner protection. For those who don't konw, SG11 used to state that "Positive corners are protected during the endgame. During the last fifteen (15) seconds of a Match, Robots may not contact Mobile Goals that are Placed in the Positive Corners of the Field…" At most of the competition we went to(particularly Essex), we would often see teams get a full mobile goal of rings before placing it in a positive corner, and guarding it until the last 15 seconds, before simply trying to score a few wall stakes or final rings, and potentially attempt a climb. I know i speak for a large part of the vex community when I say that, in most cases, the game would be over wiht about 45 seconds left, and the action-packed exciting game of high stakes would quickly turn into a simple game of just waiting in the corner, and, if your opposition had 2 full mogos, just watch preparing for your definite defeat knowing that there was nothing you could do to win.

However, this update changed all that.

Sg11 now states that positive corners become protected at 30 seconds, and although 15 seconds doesnt seem like a lot more, you have to consider that V5RC is only 1:45 long, meaning that it is almost 30% of the game where you can't get double points. This change also means that there is less time for teams to get a full mobile goal, allowing for less points to be scored before the protected period, which, in turn, means that the game becomes more competitive towards the endgame, as there is more time for goals to be scored in negative corner, which therefore leads to more intense endgames throughout the competition.

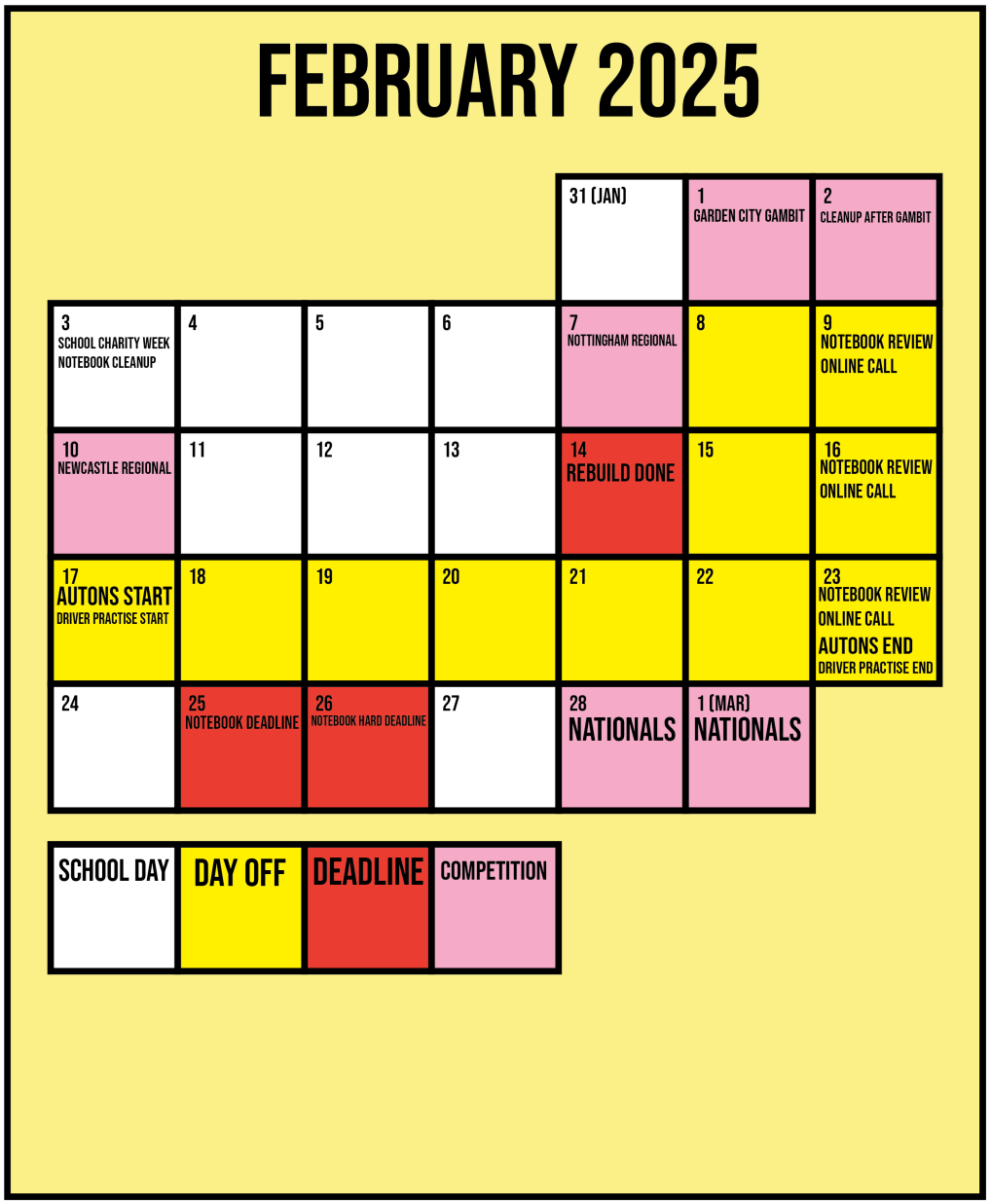
Once again, i am fully aware of how little these 15 seconds may sound, but they are so crucial throughout the match, and are a massive change in how the game is played, and in my opnion, this rule rewrite was a very good change from the GDC, and I think they have added a newer, fresher, and more competitive spin in the game.

Other rules to note:

Apart from a few revised rules; SC9 was the only other significant change in the game manual, which means that scoring the high stake is now worth 6 points, as well as the 2 points climbing bonus, as well as the likely 12 point ladder climb. This means this is now worth 20 points! which is the equivalent of just over 2 full mobile goals! I belive although we probably wont see this affect matches in the uk, i expect that in more competitive areas like America and china, we will see more robots climbing to tier 3 and scoring the high stake.

Every second counts

As of today we have today, we have under a month until Nationals. As a team we have collated all of the dates and important events until Nationals on the 28th.



As well as this plan - as discussed in Managing the Team - we communicate and meet often and in a variety of ways: in person, Discord, Trello and Whatsapp. As a final sprint effort, we have all agreed to up the amount of work, which we normally do for robotics. We have also agreed on some goals:

- Completely redesign our bot (Deadline: 14th)
- Tuning, drivers practise and autons in a good state (Deadline: 23rd)
- Cleanup the notebook (Deadline: 25th)

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae sine metu degendae praesidia firmissima. – Filium morte multavit. – Si sine causa, nollem me ab eo delectari, quod ista Platonis, Aristoteli, Theophrasti orationis ornamenta neglexerit. Nam illud quidem physici, credere aliquid esse minimum, quod profecto numquam putavisset, si a Polyaeno, familiari suo, geometrica discere maluisset quam illum etiam ipsum.

Glossary

'Auton(s)'

An abbreviation for ‘Autonomous’ or ‘Autonomous Routines’; used to abbreviate either general autonomous functions or specific routines e.g. ‘we need to code autons’

'Mogo'

An abbreviation for ‘mobile goal’; sometimes used to abbreviate a mobile goal clamp

'OU'

Abbreviation for ‘Over Under’, the 2023-4 V5RC season.

Bibliography

Bibliography

[1] V. Robotics, “High Stakes Manual.” 2024.

[2] V. F. / 'Littletimmy479', “Gearing on a six motor drive.” [Online]. Available: *https://www.vexforum.com/t/gearing-on-a-six-motor-drive/109357*

[3] BRLS, “Purdue SIGBots Wiki — wiki.purduesigbots.com.” [Online]. Available: *https://wiki.purduesigbots.com/*

[4] in_ithica | 3818, “X Drive Posting with caption: 'x drive? (wall stake mech coming soon)'.” [Online]. Available: *https://imgur.com/a/y8n1rAr*

[5] S. Magazine, “A LOOK AT HOLONOMIC LOCOMOTION.” [Online]. Available: *https://www.servomagazine.com/magazine/article/a-look-at-holonomic-locomotion*

[6] F. R. Network, “Pits & Parts | 9364H Iron Eagles - HailStorm | Over Under Robot.” [Online]. Available: *https://www.youtube.com/watch?v=EMaQuOrPwew*

[7] Wikipedia contributors, “Gear train — Wikipedia, The Free Encyclopedia.” [Online]. Available: *https://en.wikipedia.org/w/index.php?title=Gear_train&oldid=1219685098*

[8] V. F. / 'Xenon27', “Creating a Basic Drive.” [Online]. Available: *https://www.vexforum.com/t/creating-a-basic-drive/106873*

[9] S. Robotics, “How we CAD VEX Robots in Fusion 360 | Part 2 - A Simple Drivetrain.” [Online]. Available: *https://www.youtube.com/watch?v=gaJeJkWVefg*

[10] V. 98807B, “Finals Match #1, Mall of America signature | Vex High Stakes.” [Online]. Available: *https://www.youtube.com/watch?v=ixRf_hRArEQ*

[11] F. R. Network, “11101B Barcbots Getting There | Pits & Parts | High Stakes.” [Online]. Available: *https://www.youtube.com/watch?v=DbVMsuxRuag&t=396s*

[12] 6. 0470S Semicolon, “60470S 2 Piston Mogo Clamp (VEX Robotics Tipping Point).” [Online]. Available: *https://www.youtube.com/watch?v=VgsfGvBJ9-M*

[13] 2. Lucas Whiteaker, “Vex High Stakes Mobile Goal Clamp Prototype (Robot Prototype V1) | Team 23851A.” [Online]. Available: *https://www.youtube.com/watch?v=YkzFfF5XOf0*

[14] 2. 2204V Vangaurd, “22204V 1 Motor Mogo Mech (VEX Robotics Tipping Point).” [Online]. Available: *https://www.youtube.com/watch?v=Iv8quYM1e0I*

[15] V. Robotics, “Understanding Robot Features in V5RC Over Under.” [Online]. Available: *https://kb.vex.com/hc/en-us/articles/15540683424788-Understanding-Robot-Features-in-V5RC-Over-Under*

[16] V. Robotics, “V5 Motor Overview.” [Online]. Available: *https://kb.vex.com/hc/en-us/articles/360035591332-V5-Motor-Overview*

Bibliography

[17] V. Robotics, "Using the V5 GPS Sensor." [Online]. Available: *https://kb.vex.com/hc/en-us/articles/360035591332-V5-Motor-Overview*

[18] 6210K, "Comment on post: Is GPS a great tool for autonomous?." [Online]. Available: *https://www.vexforum.com/t/is-gps-a-great-tool-for-autonomous/120533/3*

[19] Mentor_335U, "Comment on post: Is GPS a great tool for autonomous?." [Online]. Available: *https://www.vexforum.com/t/is-gps-a-great-tool-for-autonomous/120533/6*

[20] V. Robotics, "Using the V5 3-Wire Optical Shaft Encoder." [Online]. Available: *https://kb.vex.com/hc/en-us/articles/360039512851-Using-the-V5-3-Wire-Optical-Shaft-Encoder*

[21] V. Robotics, "VEX Robotics Store." [Online]. Available: *https://www.vexrobotics.com/*

[22] V. Robotics, "Using the V5 3-Wire Bumper Switch v2 & Limit Switch." [Online]. Available: *https://kb.vex.com/hc/en-us/articles/360038026831-Using-the-V5-3-Wire-Bumper-Switch-v2-Limit-Switch*

[23] V. Robotics, "Using the V5 Optical Sensor." [Online]. Available: *https://kb.vex.com/hc/en-us/articles/360051005291-Using-the-V5-Optical-Sensor*

[24] V. Robotics, "Using the AI Vision Sensor with VEX V5." [Online]. Available: *https://kb.vex.com/hc/en-us/articles/24062385752212-Using-the-AI-Vision-Sensor-with-VEX-V5*

[25] V. F. / 'Ben-3176R', "Ratchet Installation." [Online]. Available: *https://www.vexforum.com/t/ratchet-installation/120206*