TODO

Table of Contents

Entries

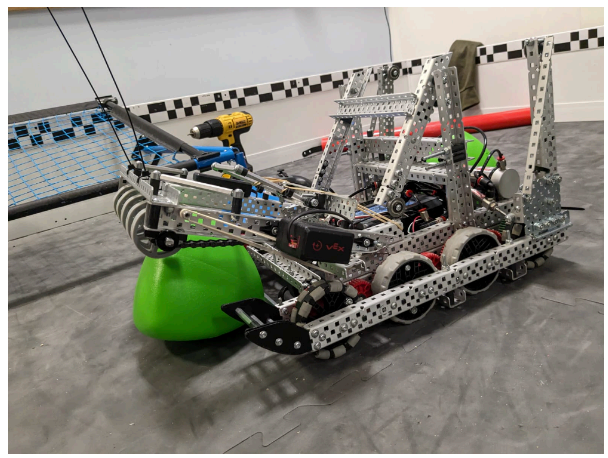
Appendix

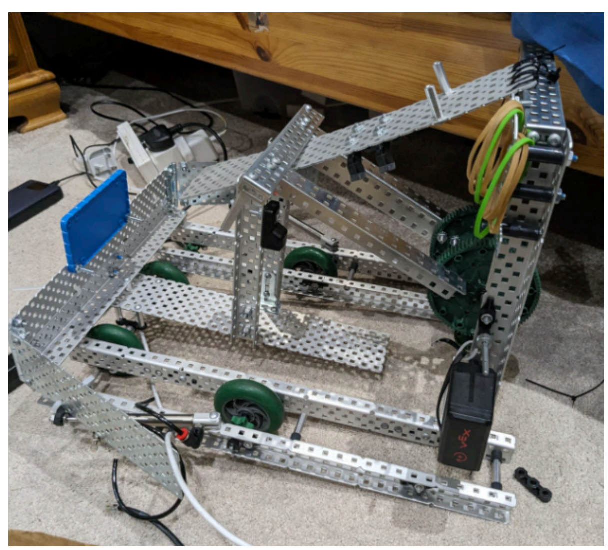
Team Introduction

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 excell 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

Jonah Fitchew

- Co-Head Builder
- 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

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.

Daniel Dew

- Head Programmer
- Driver
- Driveteam

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 DaSilva

- Programmer

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.

Team Introduction

Thomas Robb

- Head Tactics
- Driveteam

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 Enginnering 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 orginisation. The process applies to all forms of design, including programming and sometimes even tactics.

Phases of designs

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 desicision is definitely the best one available, additional proceses can be used to decide the best (i.e. descision matrices). Ideally, all members offer their thougths on the solutions.

The Snowflakes' Enginnering Ethos

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 iteratevly 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 orginised and avoid descision paralasys.

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

? Example Entry 2024/01/01

Here's some content in this entry.

Here's an example of how you'd create a pro-con table:

Pros	Cons
• pro number 1 • pro number 2 • pro number 3	• con number 1 • con number 2 • con number 3

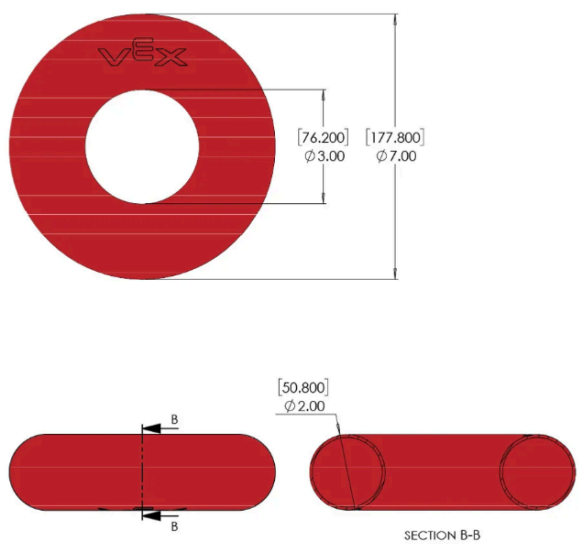
Now we'll generate 50 words of filler text!

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.

Designed by:
Witnessed by:

The Game

Scoring & Game Objectives

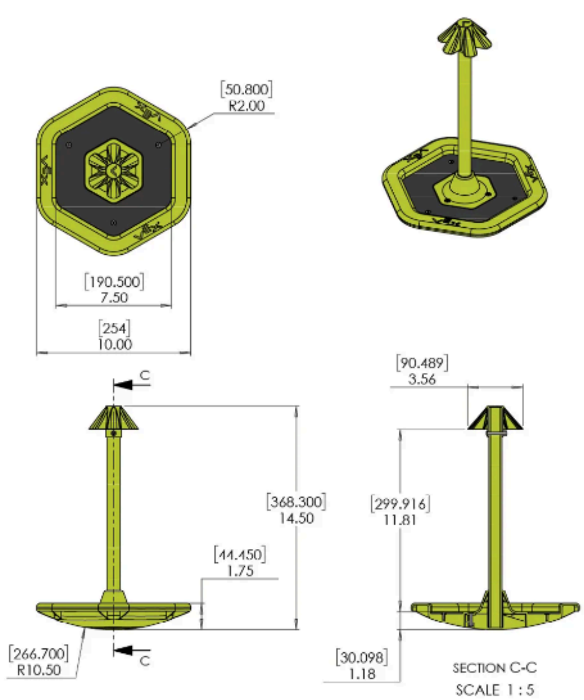


Game Elements: Rings
The majority of the scoring is done via these coloured rings.

- 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.

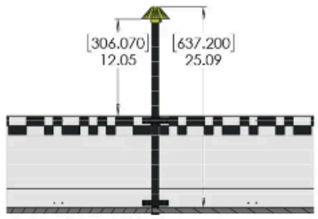
- 10'' diameter Hexagonal bird's eye view profile
- 14.5'' height
- Rubber cap to make descoring more difficult

Potential Challenges

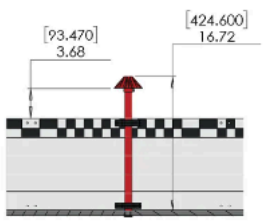
- Weighted bottom could make manipulation difficult
- Scoring would require an elevated mechanism.
- Rubber caps mean force must be required to score/descore

? The High Stakes Problem 2024/08/13

NEUTRAL STATIONARY GOALS:



ALLIANCE SPECIFIC STATIONARY GOALS:



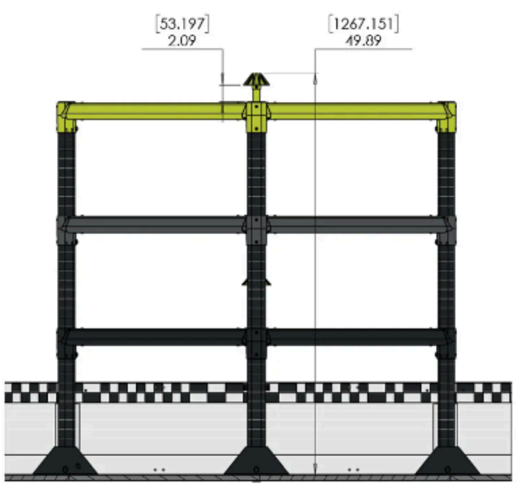
Elevated Stakes: Neutral and Alliance

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 a multiple – or one more complex, morphing – systems
- Emphasis must be put on precision and reliability as there is little room for error

Designed by: Daniel Dew
Witnessed by: Jonah Fitchew

Rules analasis

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

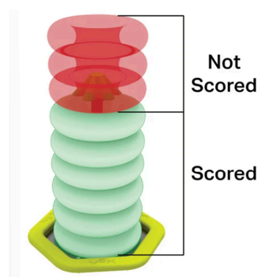
- This is one way we can mitigate the risk of infringement…
- This is another…

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

Scoring rules

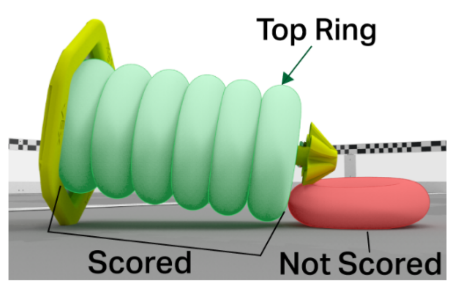
<SC3>

- To be considered scored on a stake, the ring must meet certain criteria:
  1. Ring must not be contacting robot *<u>of same alliance</u>*
  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: 6, neutral & alliance: 2, high: 1)



Problems

- Neutral/alliance stakes can contain more than 2, despite only 2 being counted
- 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

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

<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 do 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. 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 <u>cannot be removed</u>

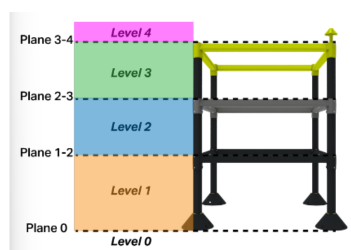
Examples included *here* .

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 independent, it cannot rely on lower rungs or the floor

Potential Solutions
- When considering climbing, large power consumption – due to independent 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
- Even if we can complete as many tasks as possible, AWP is still reliant alliance teammate, especially with no. 3

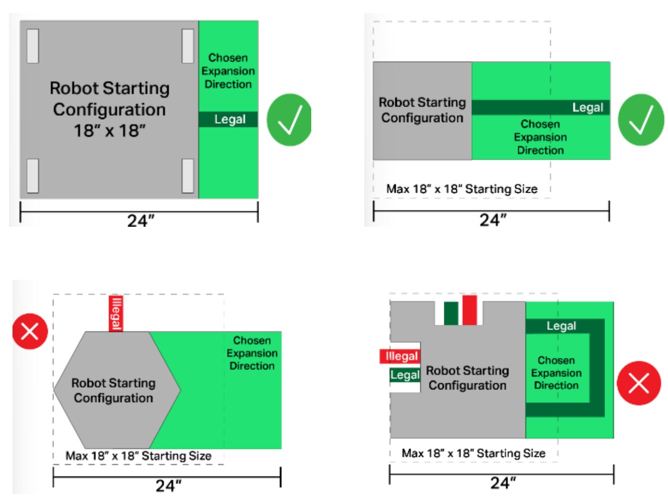
Potential Solution
- Ensure prior coordination with teammate to ensure that they move off the line at the start[3]

Specific Game Rules

<SG2>
- Horizontal expansion is limited to an additional 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.



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

Problems

- Mechanisims 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

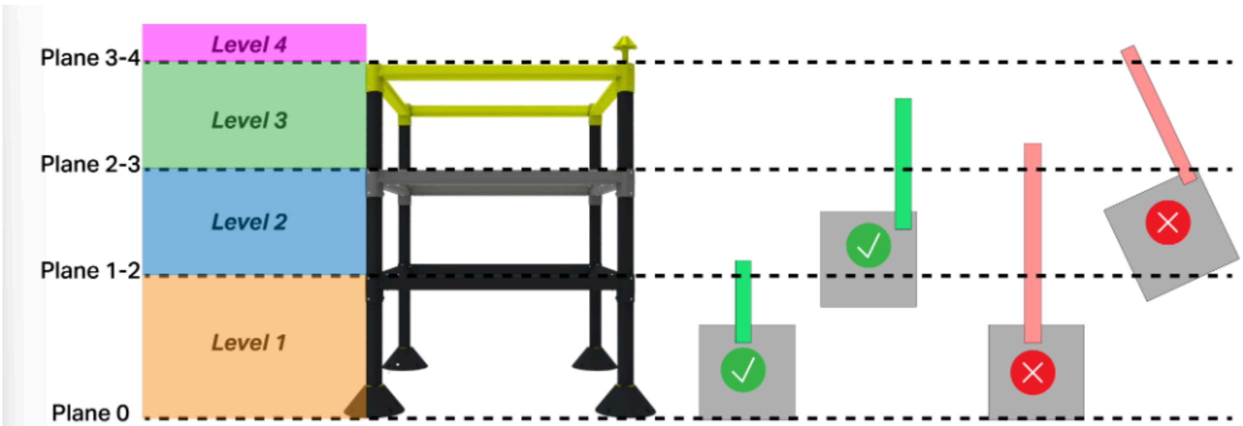


Figure SG3-1: Examples of legal and illegal vertical expansion.

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

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

Designed by: Daniel Dew
Witnessed by: Jonah Fitchew

- 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

- 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 specifically targeted 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 prescicion is key to avoiding rule infringement and also to maintain effectiveness. All mechanisms must be designed with extreme precision with lots of

[5] basically the halfway line

time allocate for fine-tuning to a) maximise effectiveness of mechanism and b) avoid breaking rules such as SG4 and SG7.

The Plan

This game and rule analasis 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

- The emphasis on driver practice means we will try our best to allow for plenty of driver practice
- The further emphasis on autonomous tuning 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 our designed with strength, prescicion and effectiveness in mind during all stages of the design process – this is especially prominent during the CAD phase

What are the different types of Drive?

The first solution to our somewhat obvious problem "We need a drivetrain" is the most conventional VEX drivetrain of them all - the differential drive, which features mirrored sides in a typical "tank" formation and allows for indepent manipulation of each side in order to achieve the movement we want to see. Second is the Omni drive which features one omni wheel in the centre of the drivetrain and allows for horizintal movement as well as the conventional forward and backwards. 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 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 manouverability - Only supports forward, backward, and turning; no lateral (sideways) movement. • Not agile - Slow to make fine, precise movements or quick direction changes

Holonomic Drives

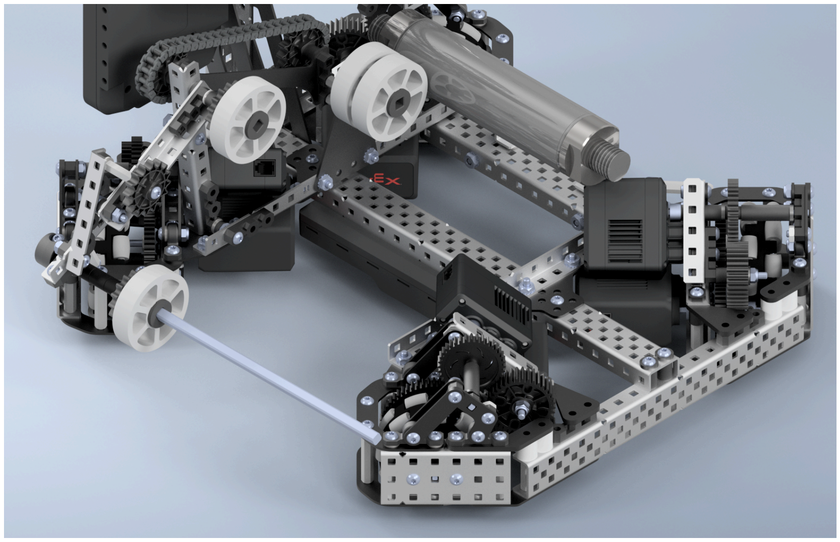
Pros	Cons
• Holonomic Movement - Can move in any direction (forward, backward, sideways, diagonal) with ease. • Manouverability - good for accurate and small adjustments • Agility - Quick, fluid directional changes without rotating the robot.	• Lower traction - Limited pushing power due to low traction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

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

Variants of Drives:

Holonomic Drivetrains

X Drive:

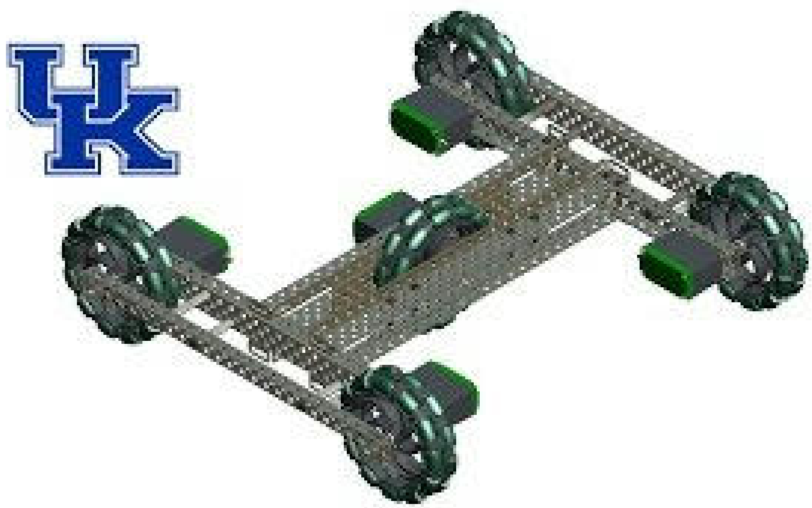


Here we see a render of an XDrive done by in_ithica | 3818 on the VEX CAD discord server

Pros	Cons
1. Higher Speed: X Drive is generally 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 2. Simplicity: Compared to Mecanum, an X Drive is simpler to build and program, as it doesn't require complex motor tuning or algorithms for strafing 3. Better Diagonal Movement: X Drive excels in diagonal motion without the power loss or inefficiencies found in Mecanum, making it smoother for navigating tight angles and corners	1. Lower Pushing Power: X Drive generally lacks traction and pushing force compared to Mecanum, making it less effective in scenarios requiring high torque or pushing resistance. 2. Inefficient Use of Space: X Drive's diagonal wheel layout takes up more space on the robot, which can limit the available area for other mechanisms compared to more compact configurations like H-Drive 3. Complex programming 4. Complex design – especially at the CAD stage

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 a more traditional competition format with the goal of Tournament Champions, X Drive does not seem like the most popular choice. 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. Packaging the brain and pneumatic tank around this drivetrain can be difficult due to the odd motor placements.

H Drive:



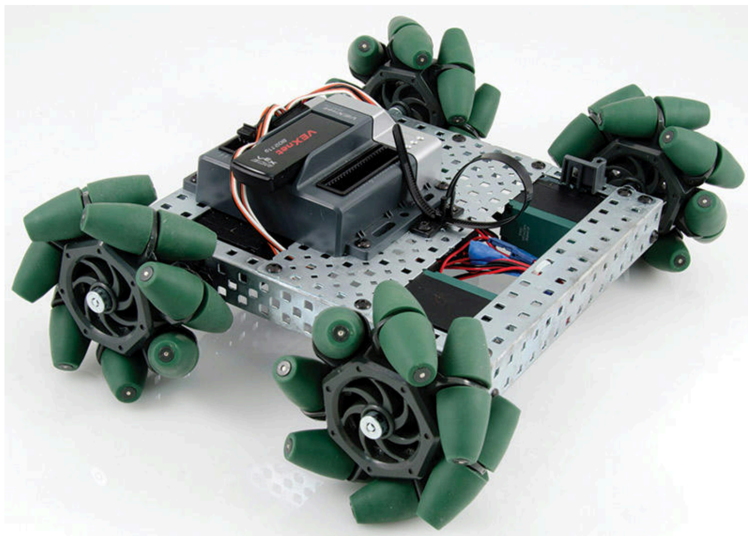
Here we see a CAD model of an H Drive found on the website Purdue Sigbots made by the University of Kentucky [1]

Pros	Cons
• Simplicity: The H Drive is easier to build and program compared to more complex holonomic drives like Mecanum or X Drive • Good Strafing Ability: The central wheel allows for decent sideways movement (strafing), providing a balance between manoeuverability and straightforward design • Efficient Use of Space: The parallel wheel layout in H Drive leaves more space for other components on the robot compared to an X Drive	• Limited Lateral Power: The 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 Agility: H Drive doesn't handle diagonal movement as smoothly as X Drive or Mecanum, reducing its manoeuverability in accurate and precise movements • Vulnerability to Central Wheel Issues: The reliance on a single central wheel for strafing means that any failure or inefficiency in that wheel significantly impacts performance • Programming Complexity: H drives also consider some (although less than X and mecanum) programming challenges • Space: Having the wheel in the middle can result in less space for other subsystems such as odometry pods

Brainstorming Drivetrain 2024/08/18

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 any traditional drive but one of the losses coming from the middle wheel is the lack of packaging ability in that area, no longer is the ability to use odometry pods to provide a higher level of accuracy during autonomous and no longer is the ability to have the pneumatic tank as low as possible to the ground due to the need for extra bracing and support of the middle wheel as inconsistencies and friction can have large effects.

Mecanum Drive:



Here is a live model of the mecanum drive found on the Servo magazine website about holonomic locomotion

Pros

- Holonomic Movement: Mecanum wheels allow for full omnidirectional movement, including forward, backward, sideways, and diagonal, giving excellent manoeuverability
- Good Pushing Power: Compared to other holonomic drives like X Drive or H Drive, Mecanum maintains relatively good traction and pushing power
- Versatility: Mecanum Drive offers a solid balance of movement options while still being able to handle various competition tasks, making it adaptable to different areas of teh competition like skills

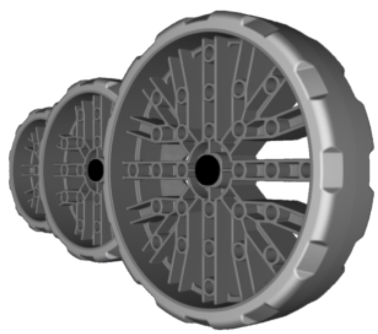
Cons

- Complexity: Mecanum drives are more complex to build and design than simpler drives like H Drive, requiring precise motor control and alignment for effective movement.
- Power Loss: Due to the angled rollers on Mecanum wheels, some power is lost during lateral movement, making it less efficient compared to a tank or X Drive
- Programming Complexity: Mecanum drives are also harder to program

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

Mecanum Drive is one of if not the most frowned upon drivetrain in the whole of VEX as it utilises the otherwise useless Mecanum wheels which take up a large amount of room and are not very versatile. however in a Mecanum drive you acquire the ability to complete control and assurance that movement is not the problem, however in building this drivetrain the ability to gear is almost completely gone, as in order to maintain the very much wanted 3-4 wide hole gap between the drivetrain C Channels the mecanum wheel leaves no room for a gear, admonishing any gear ratios wanted.

Use of Traction Wheels:



([*Image credit*])

For our drive an important consideration was the inclusion and amount of traction wheels we wanted. Having more traction wheels increases our grip but also reduces our skidding 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 in to push others.
To maximise grip whilst retaining agility, it is common to use traction wheels in the centre to reduce unwanted forces, whilst combining them with other wheels such as omniwheels on the edges for greater turning ability.

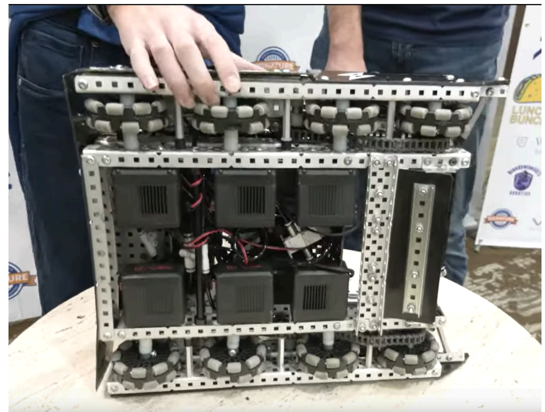
Pros	Cons
• Increased Grip: More grip allows for more power, which means its easier to push game elements or other bots around. • Resistance: Since they have a single degree of freedom traction wheels make our robot more resistant against being pushed when attacking our defending	• Rigidity: Having one degree of freedom means that the bot will struggle more to perform tight turns, which may be important to perform midmatch or during the autonomous which often relies on turns being consistent and accurate.

Overall, it is important to decide how many traction wheels we use and where. Ultimately, deciding this depends on a teams game strategy and the type of bot they are going for.
([*Citation*])

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

All Omniwheel Drive:

Having our drive consist of only omniwheels has some advantages such as allowing for a lot more skidding and tight manoeuvering. However, a lack of traction wheels means our bot have less torque to push and also will be easier to push. This pushing power loss can be compensated for by playing more evasively in order to outmanoeuver opponents. A good example of an all omniwheel drive was 9364H's bot in OU ([*Pits and Parts*]), 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.



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 power:agility ratio. Having more wheels increases surface area in contact with the ground, which correlates with more traction. An increase in traction will mean that we have less skidding, which can be useful for turning, 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 but is easier to implement since it doesn't require motor stacking.

Pros	Cons
• Agility: Slides more so it has greater theoretical turning ability • Easy to implement: Since theres 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	• Less power: Fewer wheels so there's less traction, which means less pushing power • Lighter: Less weight so its easier to push around

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

6 Wheels

6 wheels is a balance between manoeuverability and power. However, its 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 its 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 its 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, can be circumvented by quickswaps or cooling the motors with a fan.

Overall, its 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 ammount 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.

Drivetrain Selection Process

The process of selecting a drivetrain is a teams first thoughts when considering the needs for a robot in any given 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 of course, many different options are considered as the different parts that VEX provides can be combined in many different ways in order to give each drivetrain a different characteristic as it were.

Considerations for any given drivetrain

At the beginning of any season the needs of the three main aspects of a drivetrain are pitted against each other in the context of the needs of the current game, these three main features are:

- Torque
- Speed
- Size

	Manouverabili	Stability	Ease of build	Size of Wheels	Gear ratio achievable	Total
Differential	3	5	5	4	5	22
Mecanum	5	3	2	1	1	12
Omni	4	4	2	4	3	17

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 unorginised 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 aspects of conde are typically defined by some form of rule or guideline, examples include:
- Variable naming conventions
- Usage of subprograms or classes
- Choosing a language
- Version or project management
- Program structure
- Safe and secure programming

TL;DR

Standardised programming – through usage of the above – helps to improve:
- Team communication
- Debugging
- Team/project management
- Readabilty of code
- Enjoyment of writing code

Brainstorming Approaches

Here, we will define what approach(es) we will use when coding, aiming to improve useability, team communication and debugging capabilities.

Language and Software

One of the most important descisions to make is the programming langauge and its accompanying software and libraries. For robotics, the two most common langauges are C++, Python and now, Rust which is making an appearance with *vexide*[1], but the sofware 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

[1] An open source Rust runtime for v5 robots

Pros	Cons
• Python is exceptionally easy, with variable type decleration 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 • No choice in editor, therefore no choice for extensions, themes, formatting etc.

Variable Naming

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

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.)[2]

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:

Pros	Cons
• Easy to understand multiword variables • Some programs reckognise 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

[2]Dependant on language

Explanation:

- Using underlinging 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	Cons
• Very easy to understand understand multiword variables • Very easy to see where whitespace is supposed to be	• Variable names can get very long • Somewhat difficult to program with due to frequent use of ‘_’

- Boolean ‘is’ naming

Explanation:

- Start all booleans with ‘is’
- Oftentimes subprograms that return booleans start with ‘get’ (‘getIs…’)

Example:

```cpp
// (Using camelCase)
bool getIsSaturday() {
  // is it a saturday?
}
bool isSaturday = getIsSaturday();
```

Pros/Cons:

Pros	Cons
• Easy to differentiate	• Variable names can get very long • Somewhat difficult to program with due to frequent use of ‘_’

- 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 documentation harder

- Unit classification

Explanation:

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

Example:

```cpp
int getMotorSpeed() { // Dont 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

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 orginisation[3]

Pros	Cons
• Originised and readable code • Code can be run multiple times using less lines • Classes allow for even further orginisation • Classes can help mitigate developer mistakes	• Classes can take time to write • Variables defined in different files can lead to null pointers during initializing[4]

Our variable naming approach*

While variable naming conventions seem inconsequential, it is still important to decide on one and stick to it

[3]We may do a deep dive on classes at a later point

[4]Basically, the c++ compiler does not have a specified order to intiializing variables in different files, meaning if 1 variable depends on an unititialized variable (from another file) it can throw a memory error

Glossary

Example word

This is an example word which will appear in the glossary.

Bibliography

Bibliography

[1] BRLS, “Purdue SIGBots Wiki — wiki.purduesigbots.com.” 2022.