Engineering Notebook

999A

Team Number

Rising Phoenix

Team Name

Capital Robotics Club

9/1/2022	00/00/0000	1 of 1
Start Date	End Date	Book #

v1.0.8.29.22

VEX ROBOTICS

REC FOUNDATION

Resources

students.vex.com
Engineering Resources, Information on Notebooks, Videos, VEX Library, Teams Resources, and Scholarships

mentors.vex.com
Team and Mentor Resources, Mentor Professional Development, VEX Mentor Community and more

teams.vex.com
A Collection of Resources for Teams Provided by the REC Foundation

library.vex.com
Information on Building, Documentation, Troubleshooting, Coding, and other Educational Resources

About the REC Foundation

The REC Foundation's global mission is to provide educators with hands-on, student-led competition programs and educational resources to prepare future innovators for a diverse and inclusive STEM workforce. We see a future where all students design and innovate as part of a team, experience failure, persevere, and emerge confident in their ability to meet global challenges.

engineering.vex.com
notebooking.vex.com
coding.vex.com

Judging Rubric for Notebooks

vex.com
roboticseducation.org
VRC 2022-2023 Game - Rules & Game Video

Send suggestions and comments about these Digital Notebooks and Digital Parts to notebooks@vex.com

Table of Contents

About Our Team

Who Are We?

Sammy: I'm Sammy. I am one of the robot drivers. Some of my favorite activities are drawing and swimming.

Zoe: Hi! My name is Zoe Pak. I am a programmer and the backup driver for the team. In my free time, I play basketball and bike.

Vaishali: Hi! My name is Vaishali. I joined the team this year, and i'm looking forward to all the new experiences from robotics.

Vaishnavi: Hi! My name is Vaishnavi. I joined the team last year and I enjoy painting in my free time.

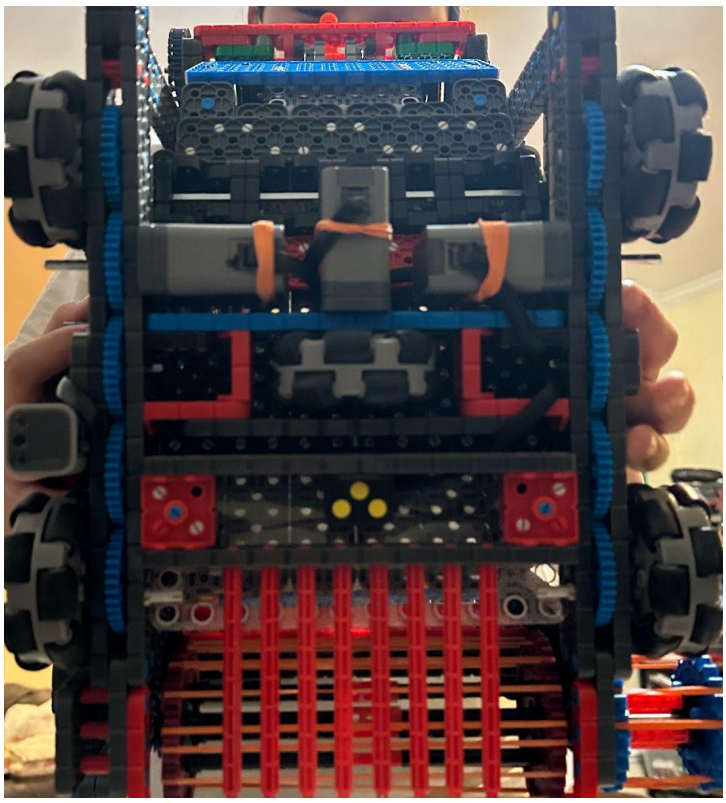
Kristen: Hi! I'm Kristen. I am one of the team drivers. Outside of robotics, I also enjoy art and dance.

Drivetrain

The H Drive

Our drivetrain is currently a H Drivetrain consisting of 5 wheels, 3 motors, and a 2x20 side length. There are two motors for each side (Left & right) which are connected with 5 gears on each side. The

Drivetrain Picture:



Shift wheel is placed in the middle of the drivetrain horizontally to help the robot move from side to side for convenience.

Project H-Drive

Name Zoe Pak Date 10/5/2022 Page 7

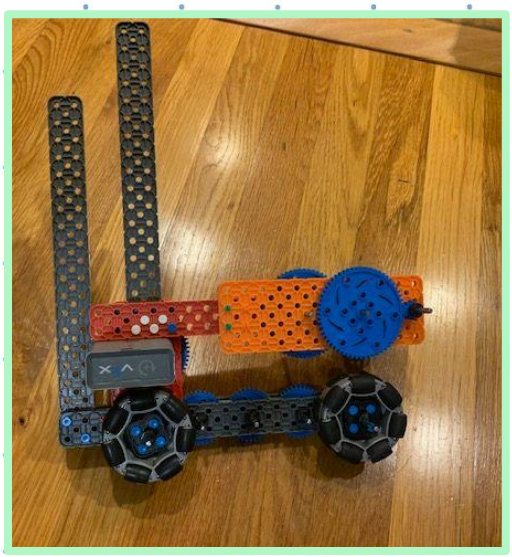
H-Drive continued...

Goal:

- Make our drivetrain a quality H-Drive to use in competitions

Problem:

A problem we encountered was to be careful, meaning we had to be very alert in order to put each small or big piece in the right position for the body of the robot to fit.

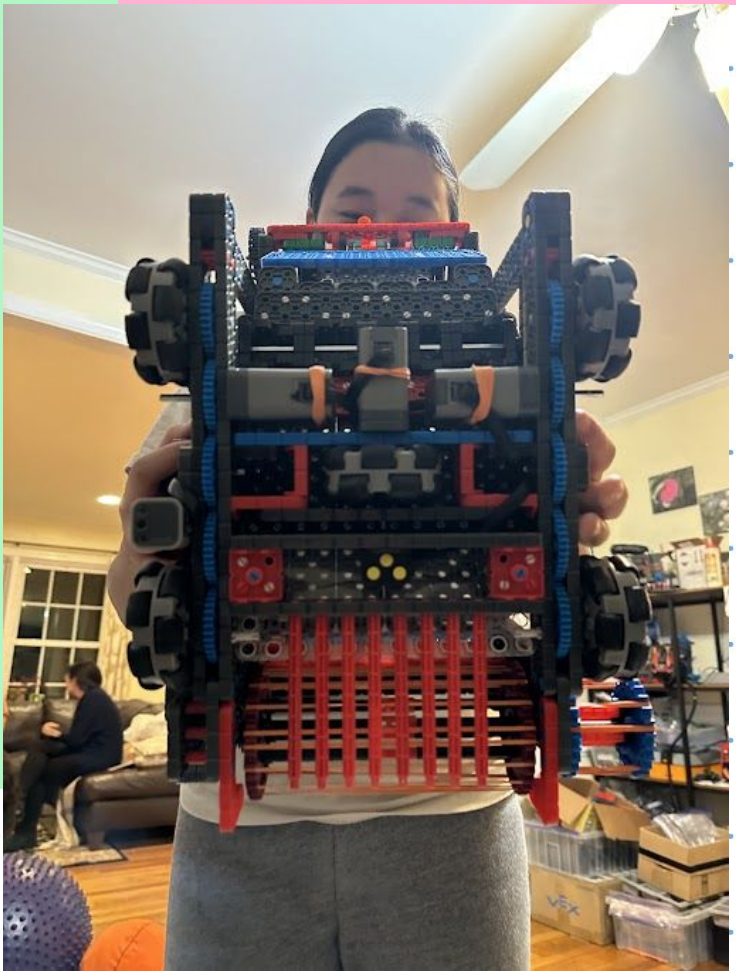


SHIFT WHEEL

How does the shift wheel help us?

- The shift wheel allows us to move side to side when shooting as well as when extending to gain points
- It allows us to adjust the robot if it is off target when trying to unload the dispensers
  - Particularly helpful when controlling the yellow dispenser

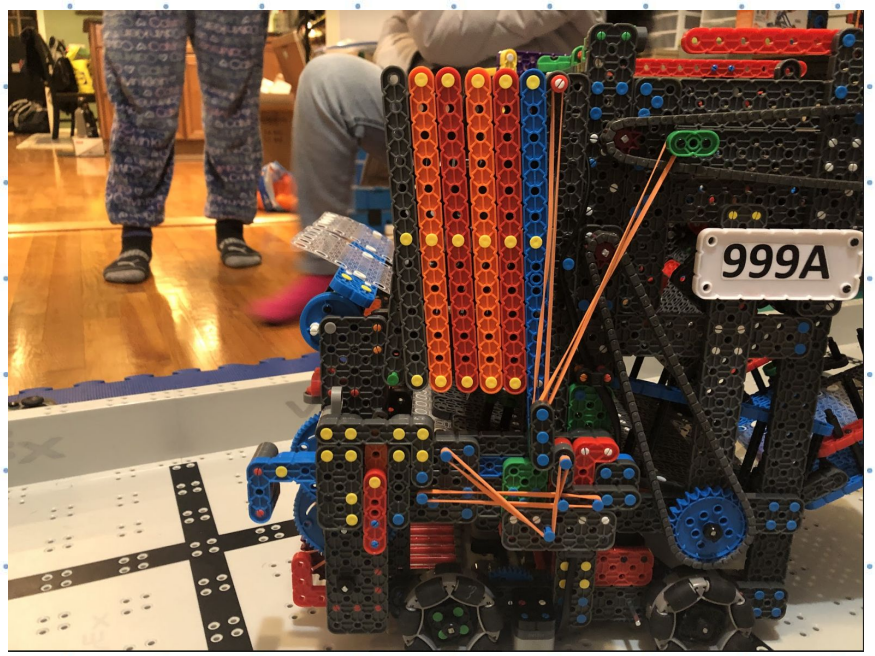
The shift wheel mechanism allows the robot to go side-to-side. We made this design to help us move our robot with more ease. We decided that this would make our robot easier to control and therefore more efficient and effective.



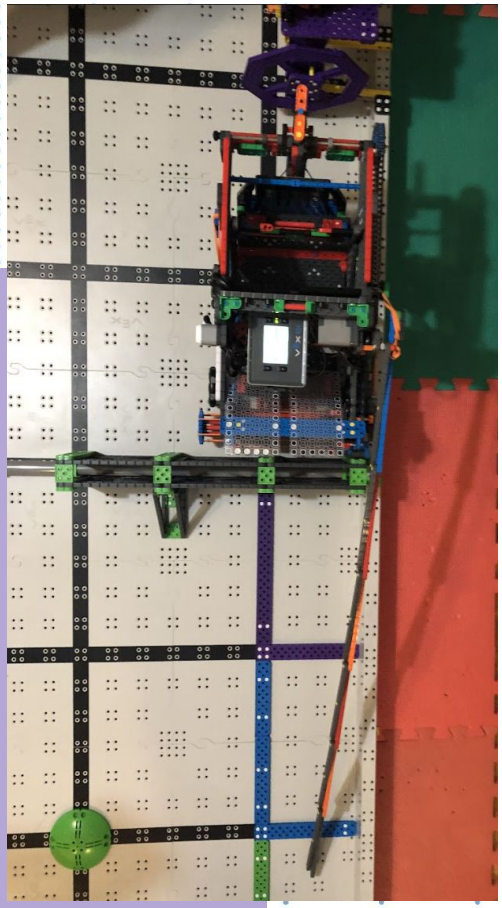
Project Shift Wheel

Name vaishnavi Date 1/13 Page 9

Shift Wheel and Touchdown Motor Trade off



we had one motor that we could use, either for the shift wheel and touchdown motor. we had to make a tradeoff and decided to use it for the shift wheel.



we found how to create a touchdown mechanism that operates without a motor as we did not want to give up our shift wheel. As stated in the previous slide, the mechanism will project out onto the green portion of the board when the small red "L" piece makes contact with the wall of the divider. The connected pieces are moved out of the way and the arm is released when the L piece is struck. The arm extends and lands as a result of the rubber bands' tension.

Gears Instead of Chains?

Usually, we will have two wheels on each side of the drivetrain that are connected with a chain. But, last we discovered some problems: the wheels wouldn't drive straight. The wheels were very hard to control. So, after some observing, we learned that most other robots used gears to connect the two wheels (as you can see in the photo above). This year, we are testing to see if gears will be more reliable for us, so that it will be easier to drive and program the robot. Our hypothesis is that the gears will be more stable than the chain.

Project Why we switched to gears for drivetrain

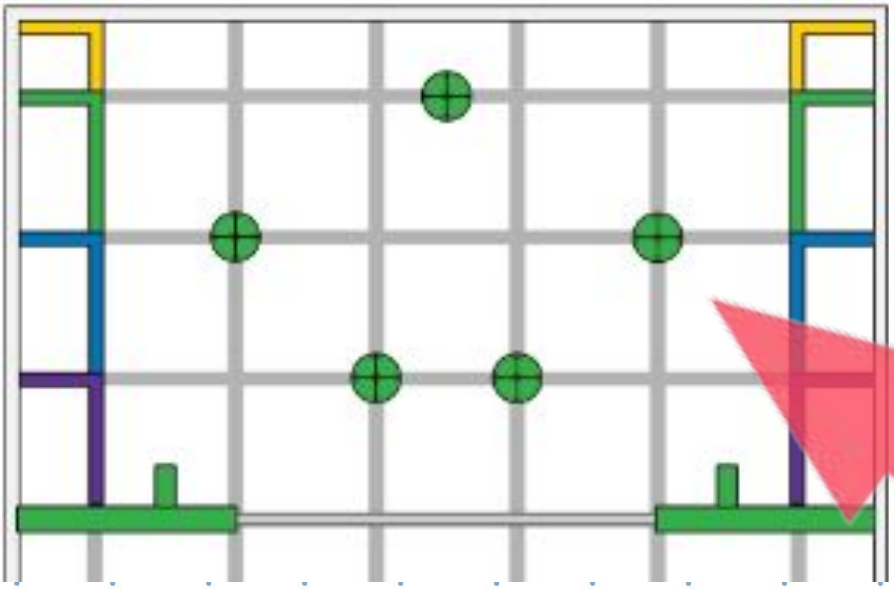
Name Kristen Date 10/11 Page 11

Shooter

Our Goal!

When we started, we knew we wanted a shooter that was efficient; it should be...

- easy to load
- easy to shoot
- hoot all pucks quickly.

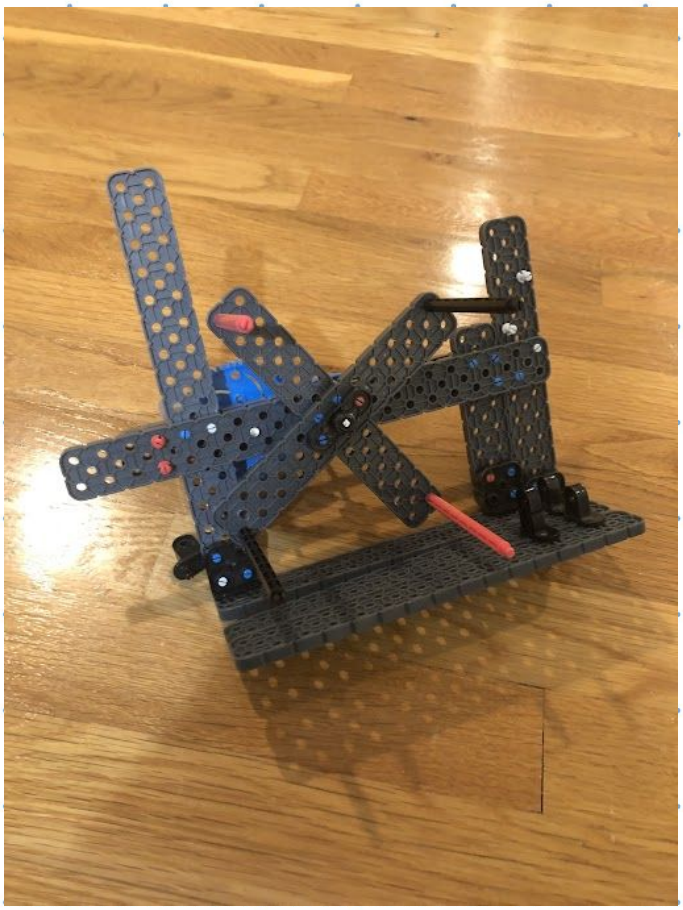


We concluded that having a double shooter would help us meet these goals. This is because it's more efficient than one, and anything more than 2 would be too wide to fit between the green stoppers in the scoring zone.

To ensure the shooter is easy to load, we decided that there needed to be a wall around the shooter, so that the pucks didn't fall out. There needed to be a "ceiling" above it too to direct where the pucks will fall. We needed the pucks to fall flat into the shooters.

To figure out how many pucks go into each shooter, we did some math! There are 45 pucks, and most of them would stay on the ramp when there isn't space. We predicted that each shooter would hold 8-10 pucks.

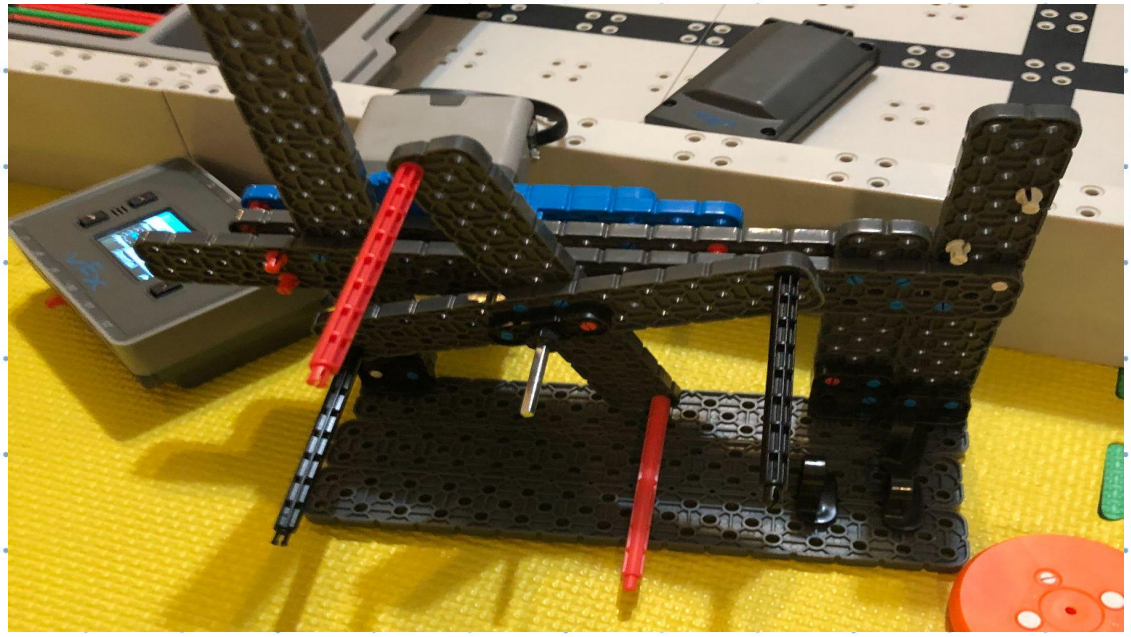
The Ferris Wheel Design



During some of our first meets, we began to brainstorm some ideas for shooters. This design, that we named "the ferris wheel", was one of them. A motor is attached to a gear (gear ratio was 5:1), which is attached to the section that looks like a cross. The standoffs connected to the ends of the cross will hit the pucks. While the motor spins, the standoffs will also spin. The pucks get...

Project Beginning to design shooters

Ferris Wheel Continued



...knocked out when the standoffs spin. Since there are four standoffs, four pucks will be shot very quickly.

Why this design didn't work:

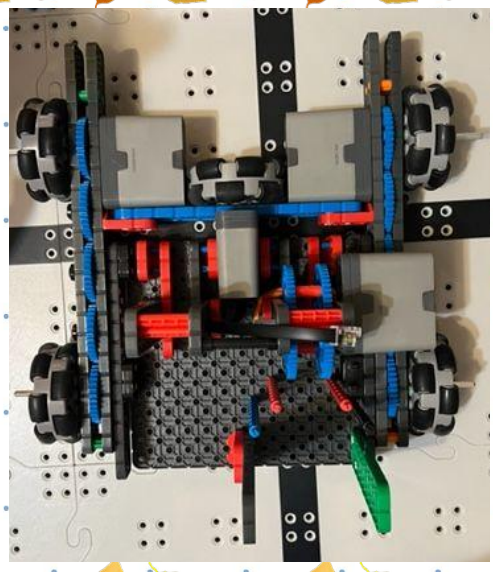
- It was difficult to find a way to load pucks into the mechanism.
- The standoffs didn't have enough power to push the pucks far enough.

The Vending Machine Shooter:

The vending machine shooter is a simple mechanism that mainly consists of a single 2x beam, that repetitively goes back and forth shooting one puck at a time. There are two slots for two piles of pucks (Preferably 8 pucks each) and one shooter for each pile. The 2x beam would shoot the puck closest to the bottom until the stack is gone.

Problem:

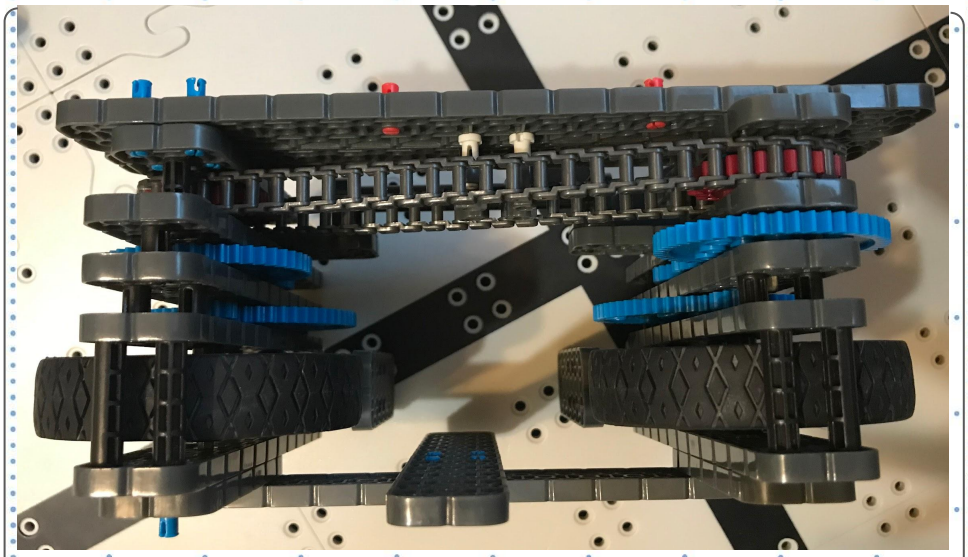
A problem we came across was that we had to retract the 2x shooting beam in order for the pucks to be evenly stacked on top of each other. As tested with the beam underneath, the first puck would be slanted and would cause the pucks to be slanted; which would cause them to slide out of their position.

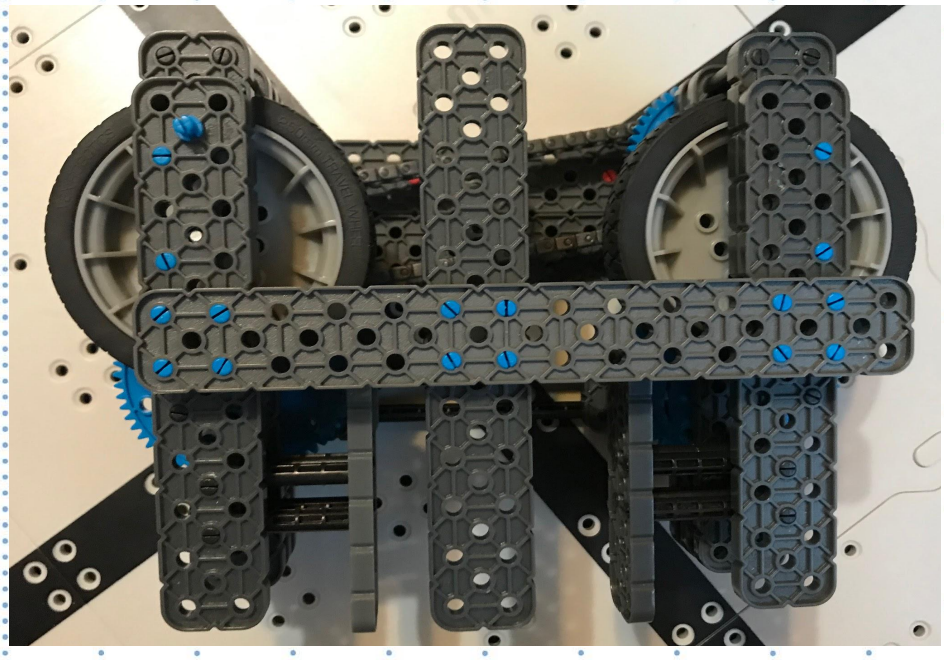


Solution:

The solution we decided to use was a rubber band retracting system which would stretch the beam back allowing more space for pucks to stack on top of each other.

The Flywheel Shooter





Overview:

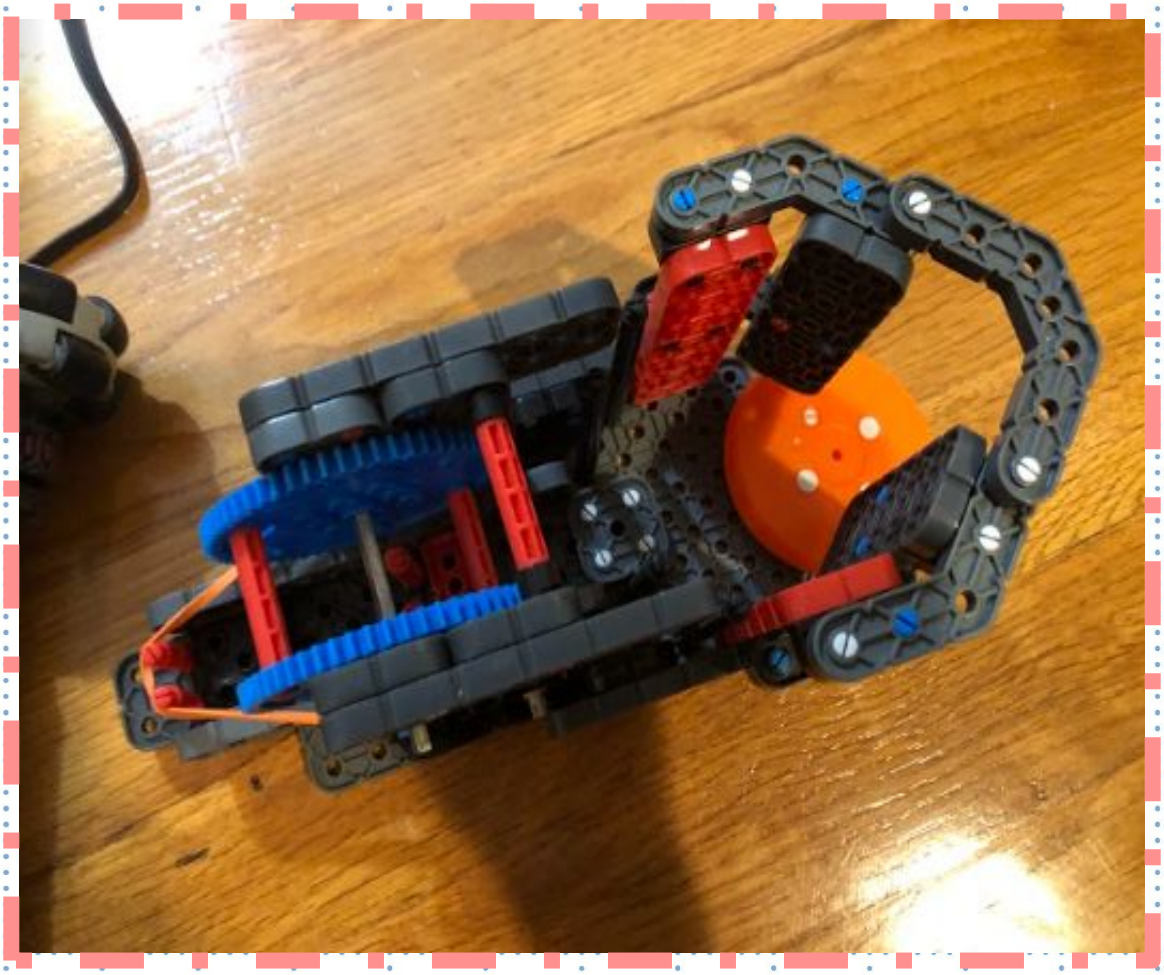
The flywheel shooter is composed mainly of two wheels. The wheels are spun and disks can be shot out of it. Since there is a 6 motor limit, then the flywheel is extremely clunky and complicated.

Why not?

The reason we decided to not go with the flywheel is that it took up to much space. There was also a smaller, simpler design we had.

Pro's and Con's

The main pro to this design is that it is extremely efficient at shooting out disks. The main con is that it is very large and takes up a lot of space on the robot.

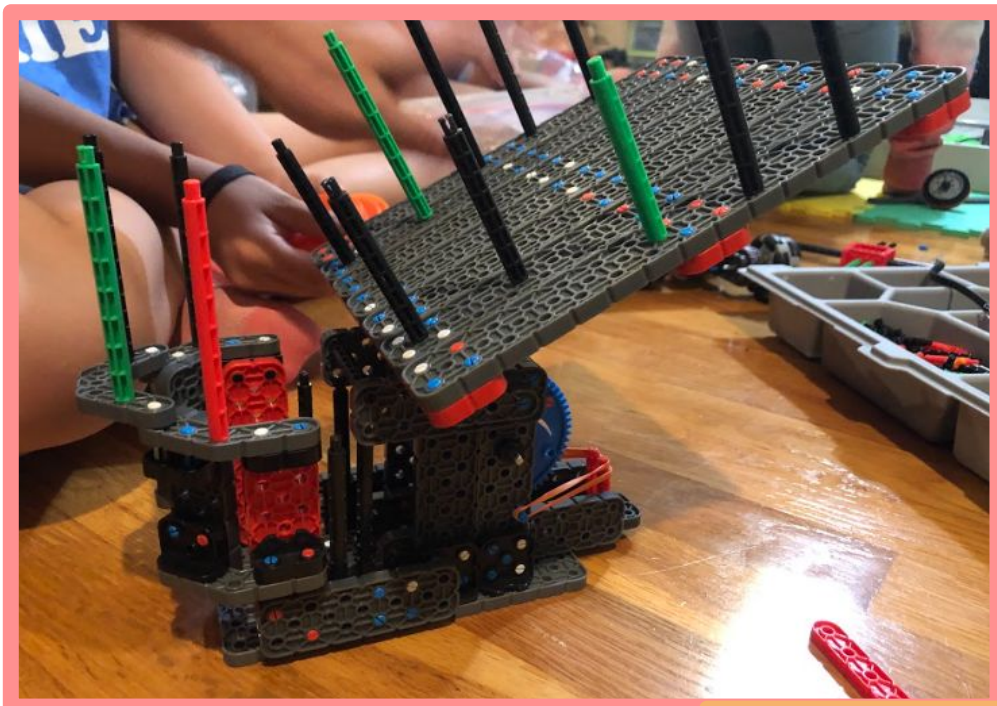


New Design!

This mechanism has a gear that pulls back and releases a bar. The bar then hits the pucks and propels it really far. The gear is attached to a motor, and it turns really fast.

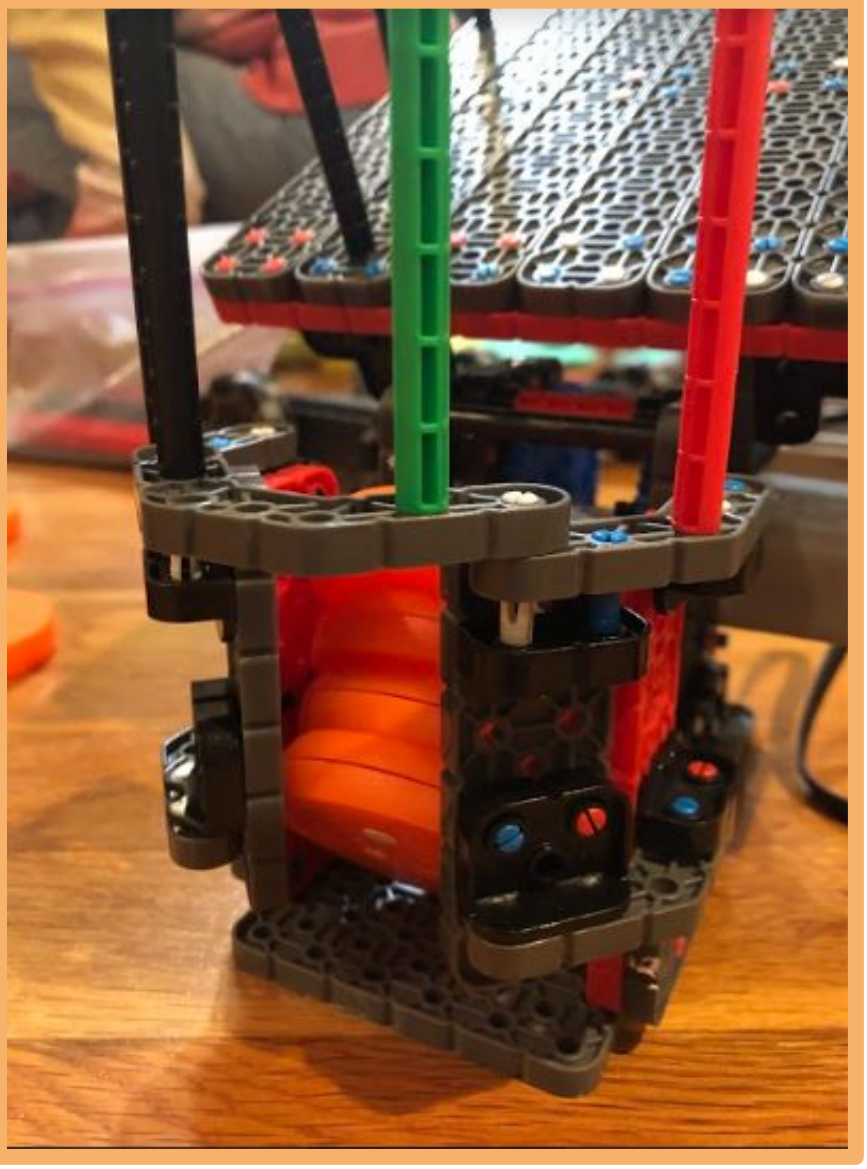
Pros:

- Shoots pucks out really fast-very efficient
- compact design-and easily be changed to a double shooter.



After trying the new design, and learning that it was successful, we attached the ramp we designed last week onto it. The ramp is attached so that pucks can easily slide into the shooter.

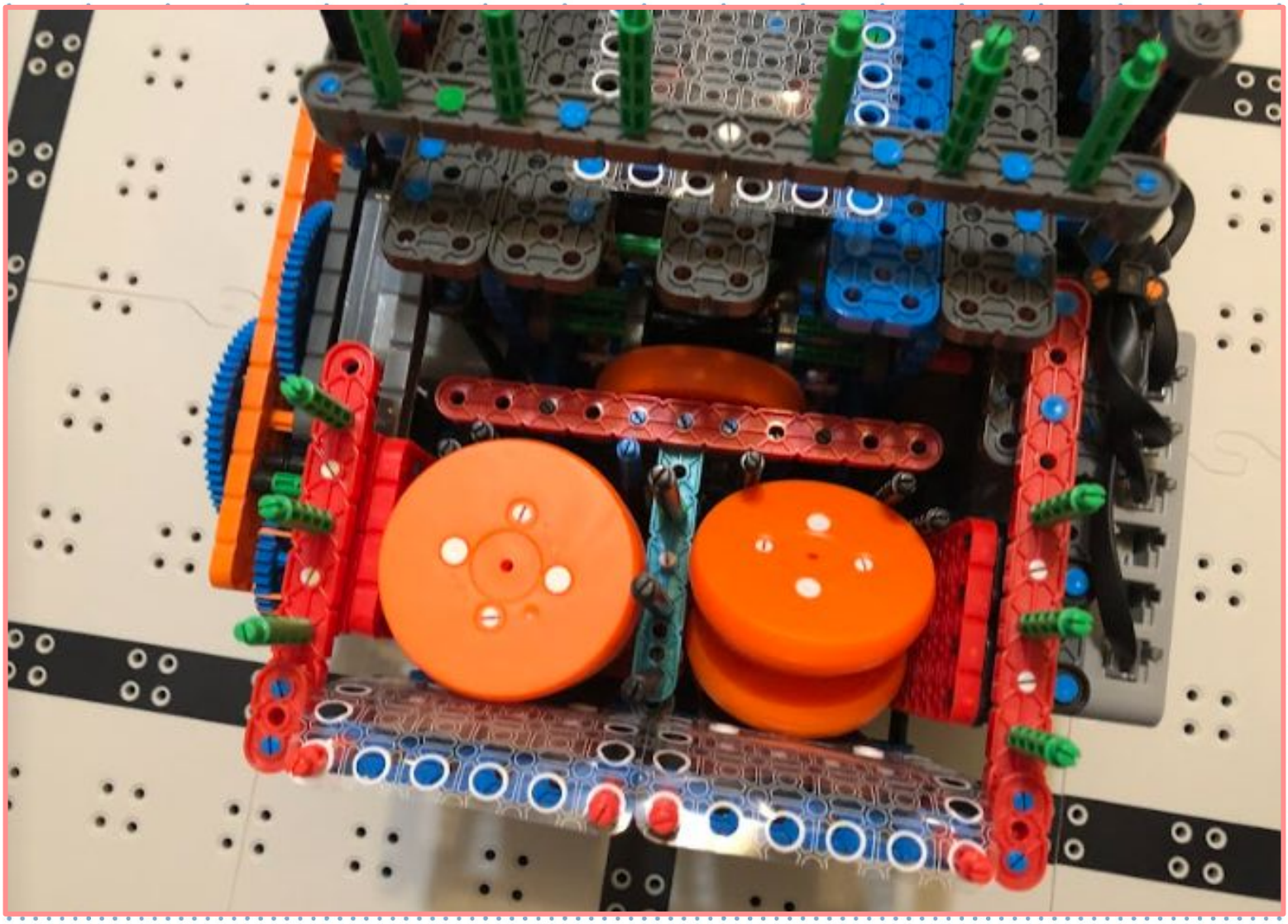
when we rolled the pucks in, we start discovering new problems. As you can see, the pucks wouldn't fall into the shooter flat, because the bar that is supposed to hit the pucks is rested right under the landing spot. This would mess up the whole shooting process, because the bar wouldn't be able to hit them as far. We needed to figure out a way to move the shooter bar out of the way



Project Shooter Design Continued

Name kristen Date 10/3/22

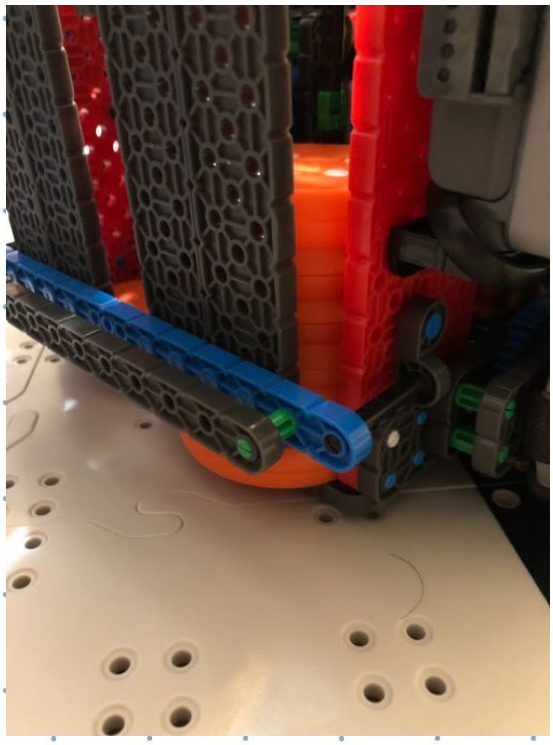
Problems We Encountered in Our Design: Part 1



The first problem that we noticed with our design is that the pucks don't fall into the shooter correctly. As you can see, in the right chamber the pucks fall in sideways, and not completely flat. When the pucks are sideways, the shooter isn't able to reach them to shoot, meaning we can't score any points. Sometimes, the pucks will also fall into the inside of the robot (as shown above). This is a problem because it clogs up the shooter, and there is no way of getting it out. Also, the middle section that divide the two chambers are not very sturdy. We plan to fix all of these problems so that we are more likely to reach our goal.

Project Problems with the current shooter

Name kristen Date 11/25



Problems We Encountered in Our Design: Part 2

on the first day of driving practice, we noticed some problems. Assuming that the pucks do fall into the shooter correctly, sometimes they will slightly slide out. when this happens, the shooter isn't able to hit the pucks, since the distance is too far. when the pucks slide slightly out, sometimes they will completely fall out after driving for a while. we plan to fix this by adding some sort of grip so that the pucks and the beams have more friction against each other.

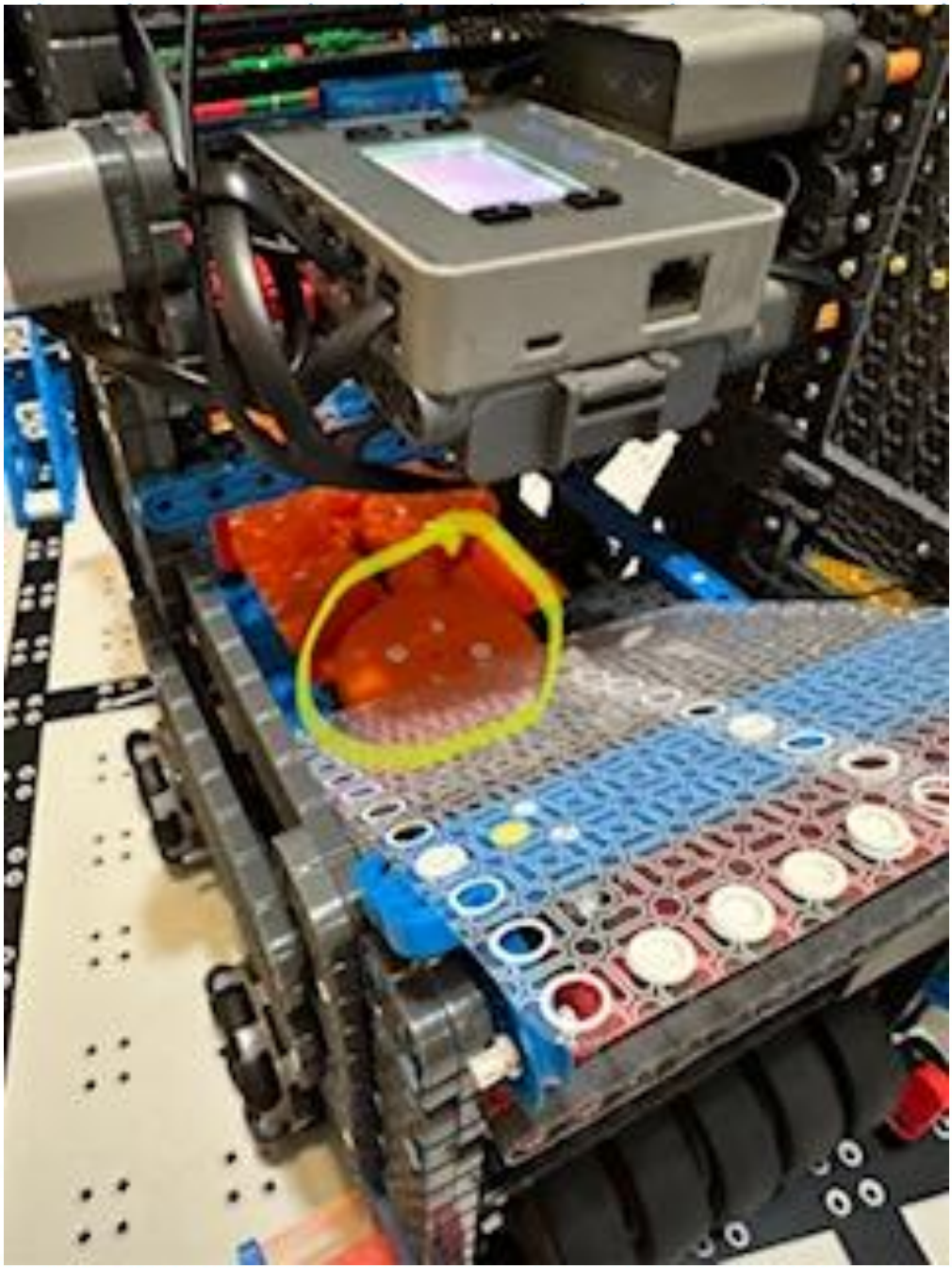
Project Other Problems We Encountered

Name kristen Date 11/25 Page 21

Problems We Encountered

Part 3:

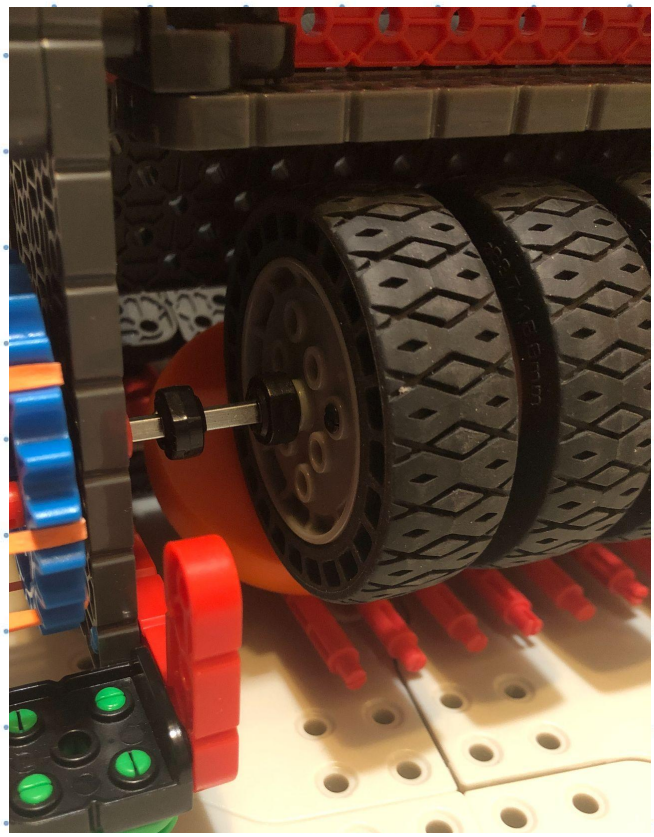
To the left, this photo shows an orange puck getting stuck on top of a plastic piece where it was not supposed to be. This caused the shooter to get jammed and the ramp could not move to shoot.

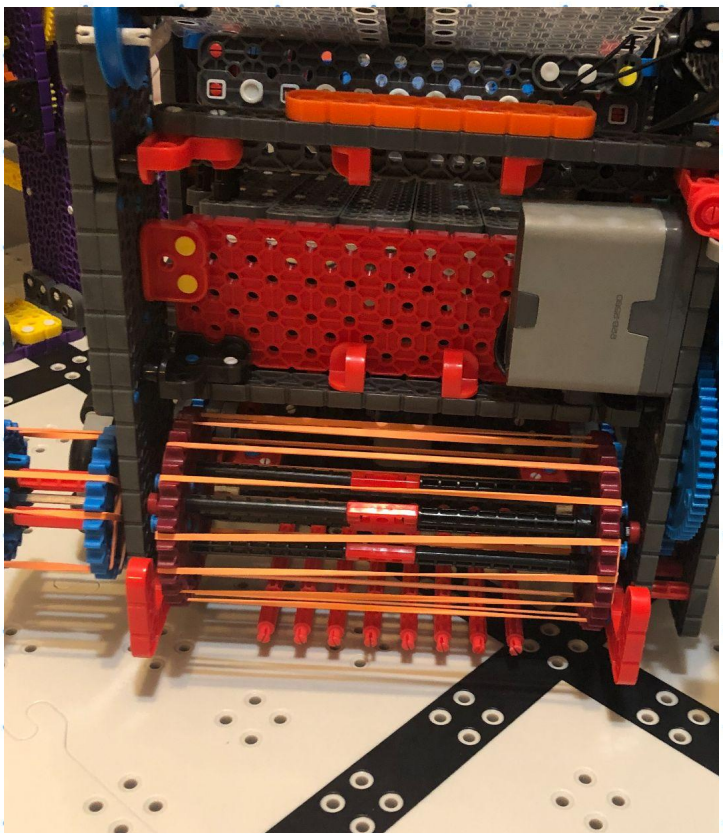


A Better Shooter Design!

we changed this shooter design because of many reasons, but mainly because of the inconsistency of the shooting. In the next few pages, we will go over each reason why the new design with the rubber bands is much better than the one with wheels. In the future, we also plan to change this design to be suitable for collecting pucks too, just like how we collected balls in Pitching In last year.

Before ..... After!





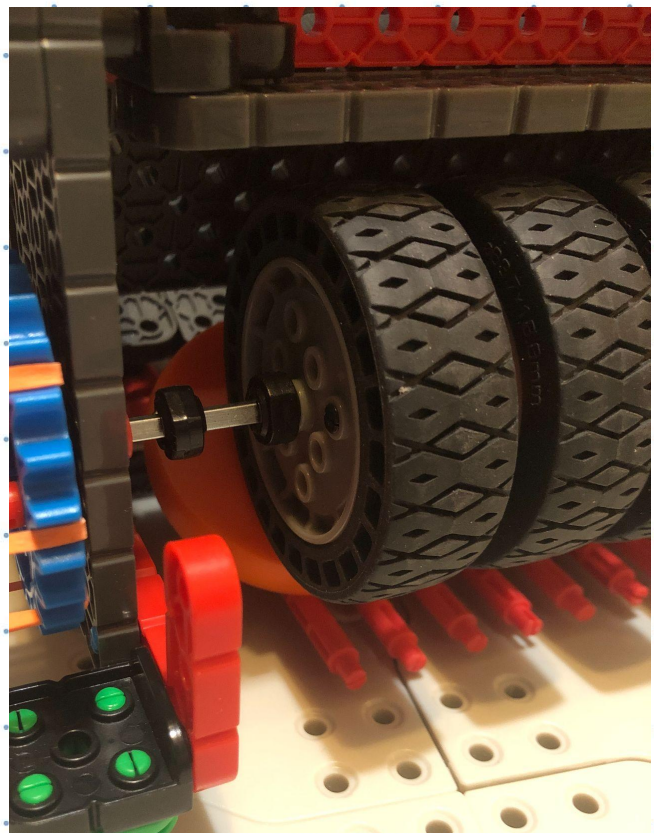
Project A better shooter design

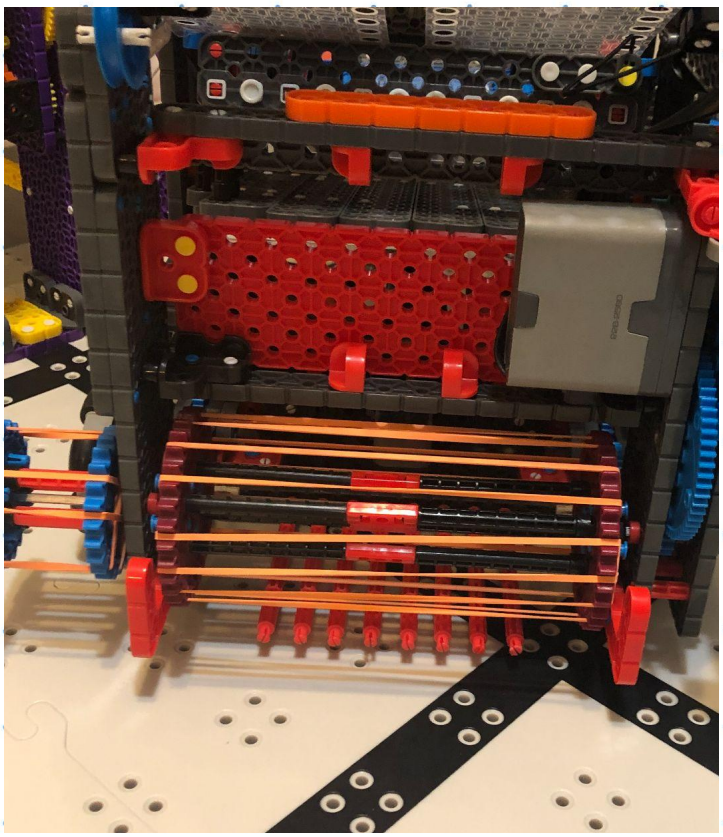
Name kristen Date 1/14

A Better Shooter Design! Benefit #1

changing the shooter from wheels to rubber bands was a big benefit, because it is lighter. Before, it would take around 3 seconds for the wheels to start spinning and accelerating to full speed due to its heaviness. Now, it immediately goes to full speed because it is lighter. Since we only have one minute per run, every second counts.

Before ..... After!



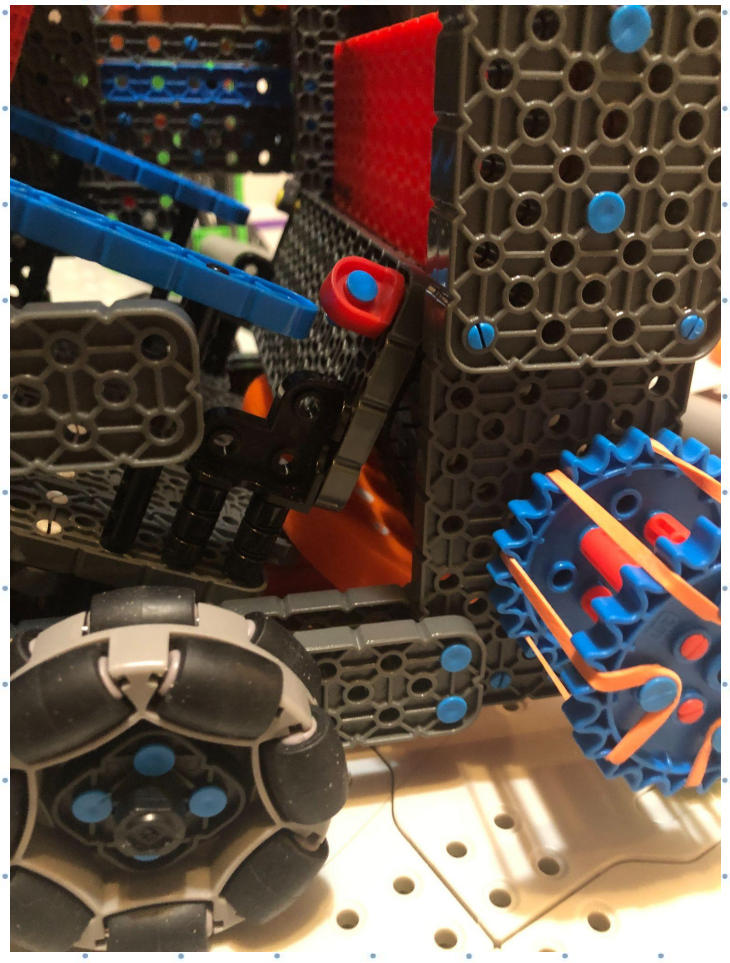


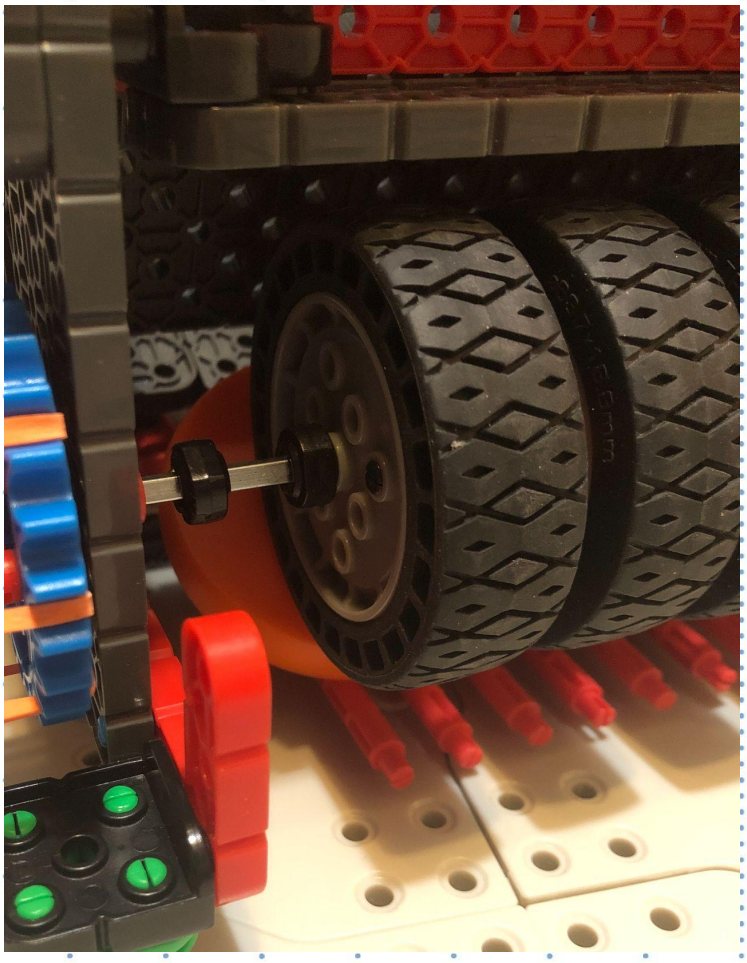
Project A better shooter

Name kristen Date 1/14

A Better Shooter Design! Benefit #2

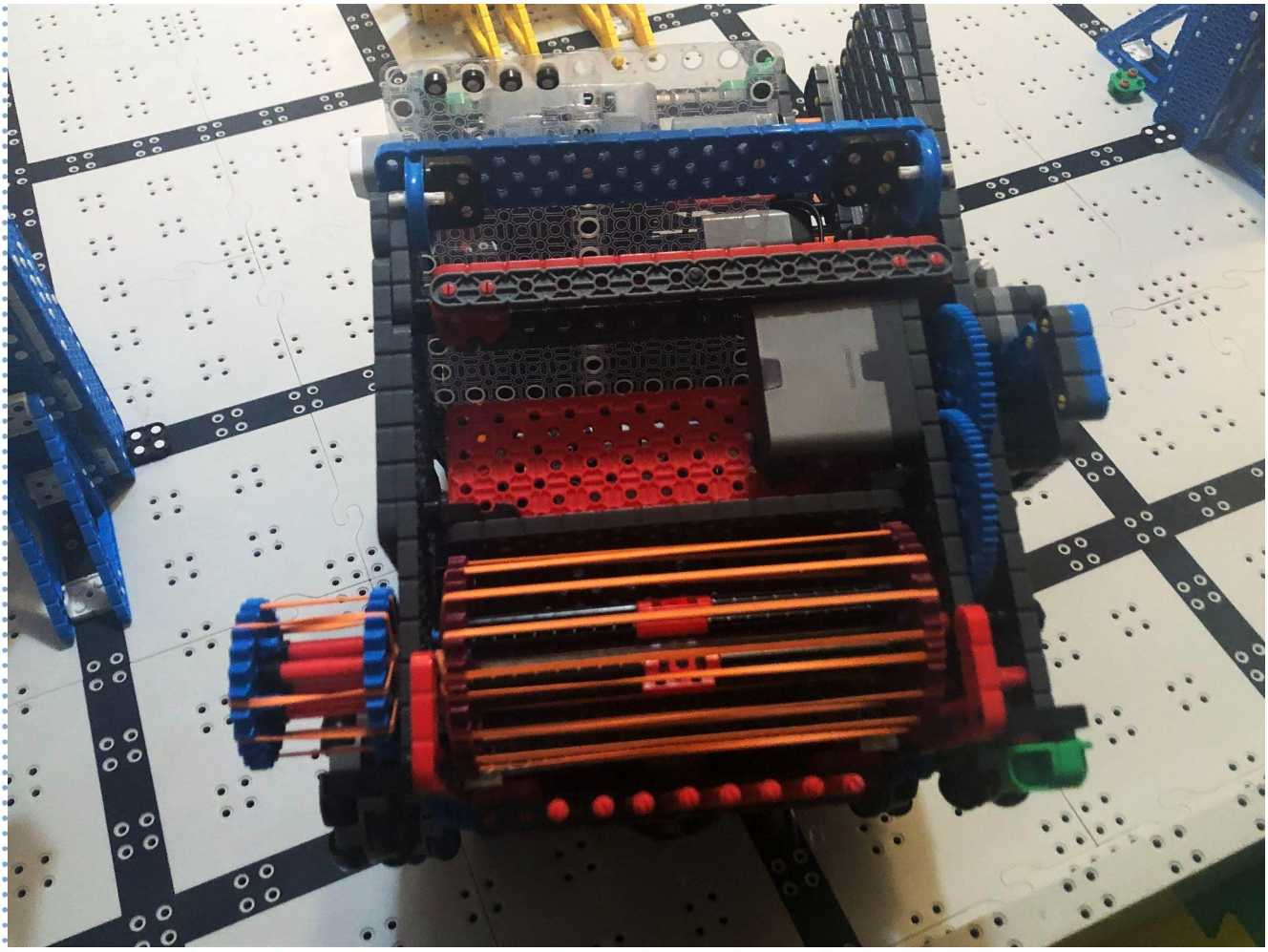
As you can see in the pictures below, the pucks consistently got suck under or over the wheels. Once that happened, it wouldn't allow any other pucks to go through, making the whole robot un-functional. Since the rubber bands were more flexible, the pucks would never get stuck. The pucks would stretch the rubber bands and always make its way through the robot.





A Better Shooter Design! Benefit #3

The roller for the purple tower is made of rubber bands, and the shooter is made of rubber bands. After some thought, we realised that we could use the shooter as a roller too (more information on slide 45). With the wheels, it was hard to line up the small roller with the tower. But now, we have a bigger surface when using the shooter.

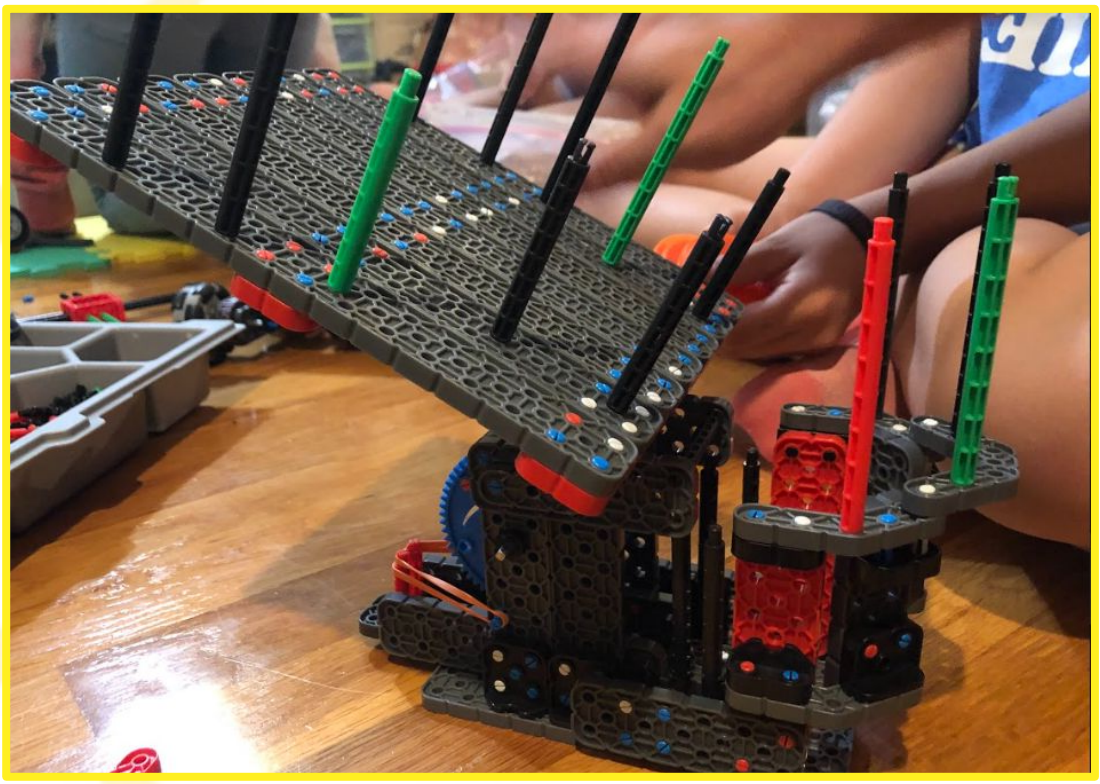


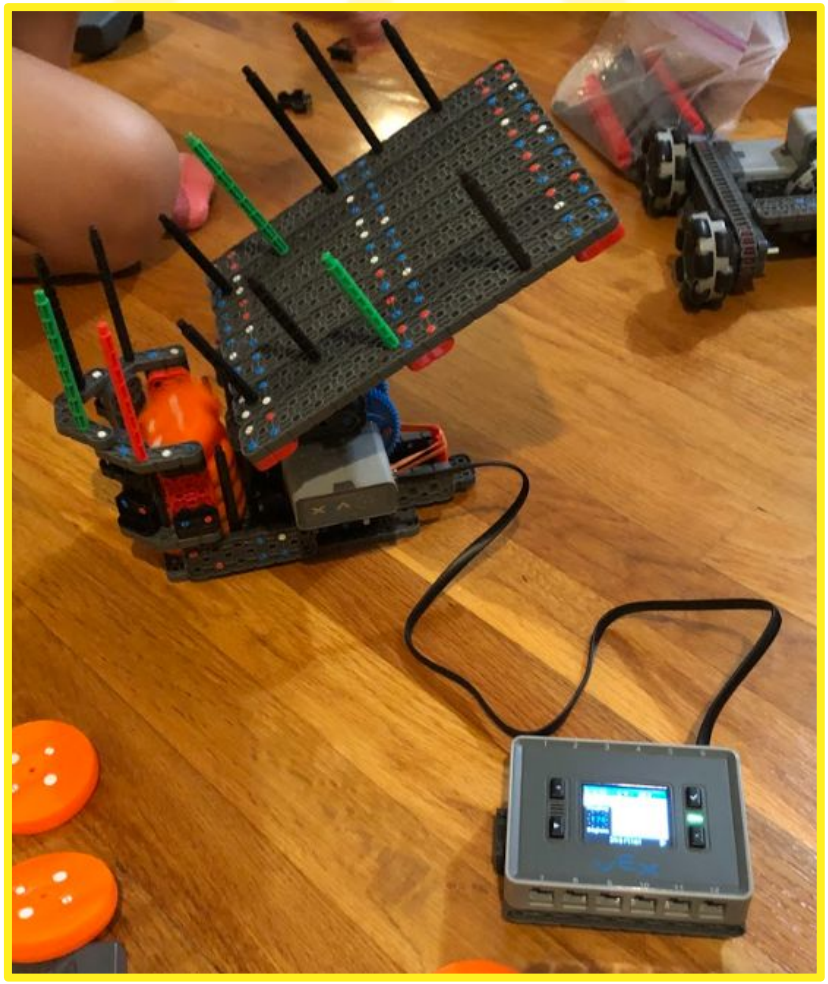
Project A better shooter

Name kristen Date 1/14

Loader/Ramp

First Ramp Design





This design was made up of many 2x bars. The stand offs were added to lead the pucks into the shooter. We were planning to have the pucks drop onto the ramp, then roll into the shooter. There were a few problems with this design:

- It was heavy/bulky
- The pucks usually didn't roll into the shooter properly (see page 17).
- The angle of the ramp wasn't adjustable

Conveyor Belt Design

This design picked up pucks and then transported them to the ramp by using pulleys that looped around. This designs goal was to efficiently and easily pick up multiple pucks at a time to deliver to the ramp. Although, there were a few problems with the design.

PROBLEMS

- It was hard to build around the design to protect the pucks at the sides (from falling off)
- Large
- Hard time picking up pucks

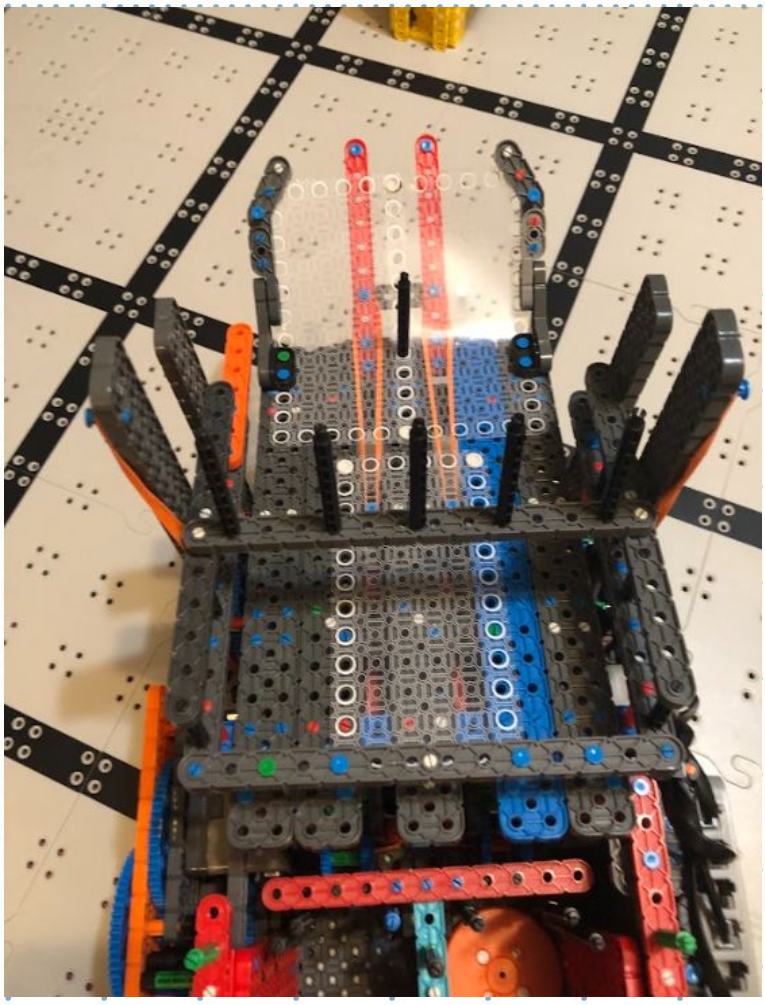
Conveyor Belt Design Pt.2

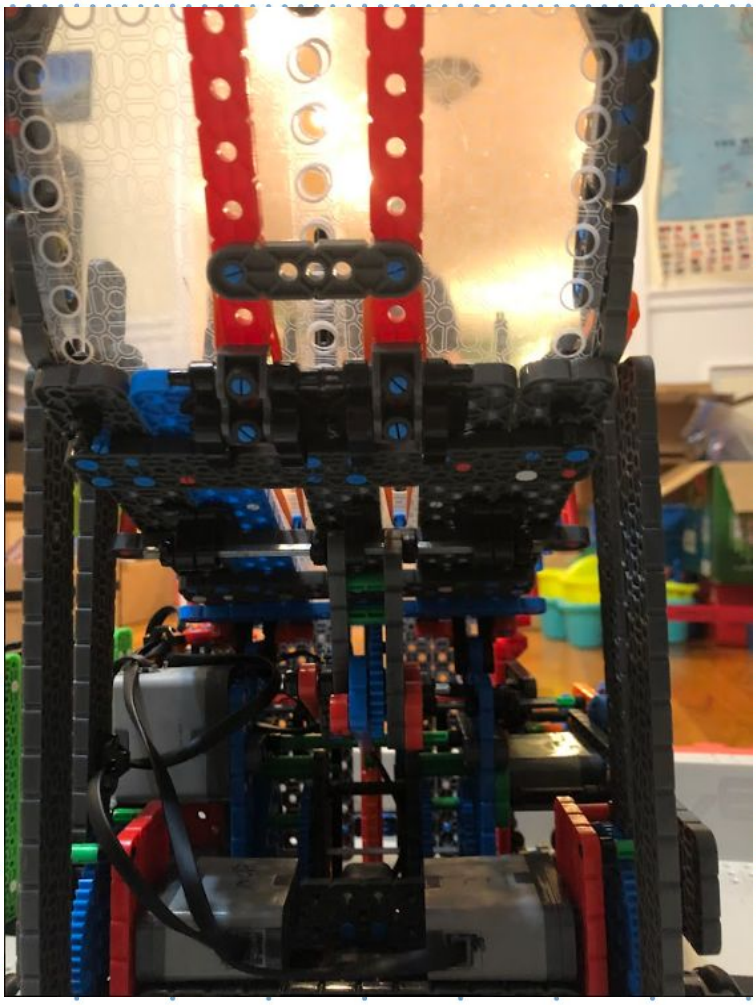
The primary explanation we decided against implementing the design were related to some technical issues with the edges as well as its overall bulk and shape. There was nowhere to set it up that would function well. The pucks wouldn't be able to enter if we blocked off the sides.

PROBLEMS

- Hard time fitting the pucks into the conveyor belt
- Difficulty adjusting the sides of the conveyor belt, and managing their sizes
- A problem occurred when looking at the sides of the conveyor belt, the pucks fell off at the side.
- There was no room for it; non-efficient.
- slow

Sturdier Ramp Idea





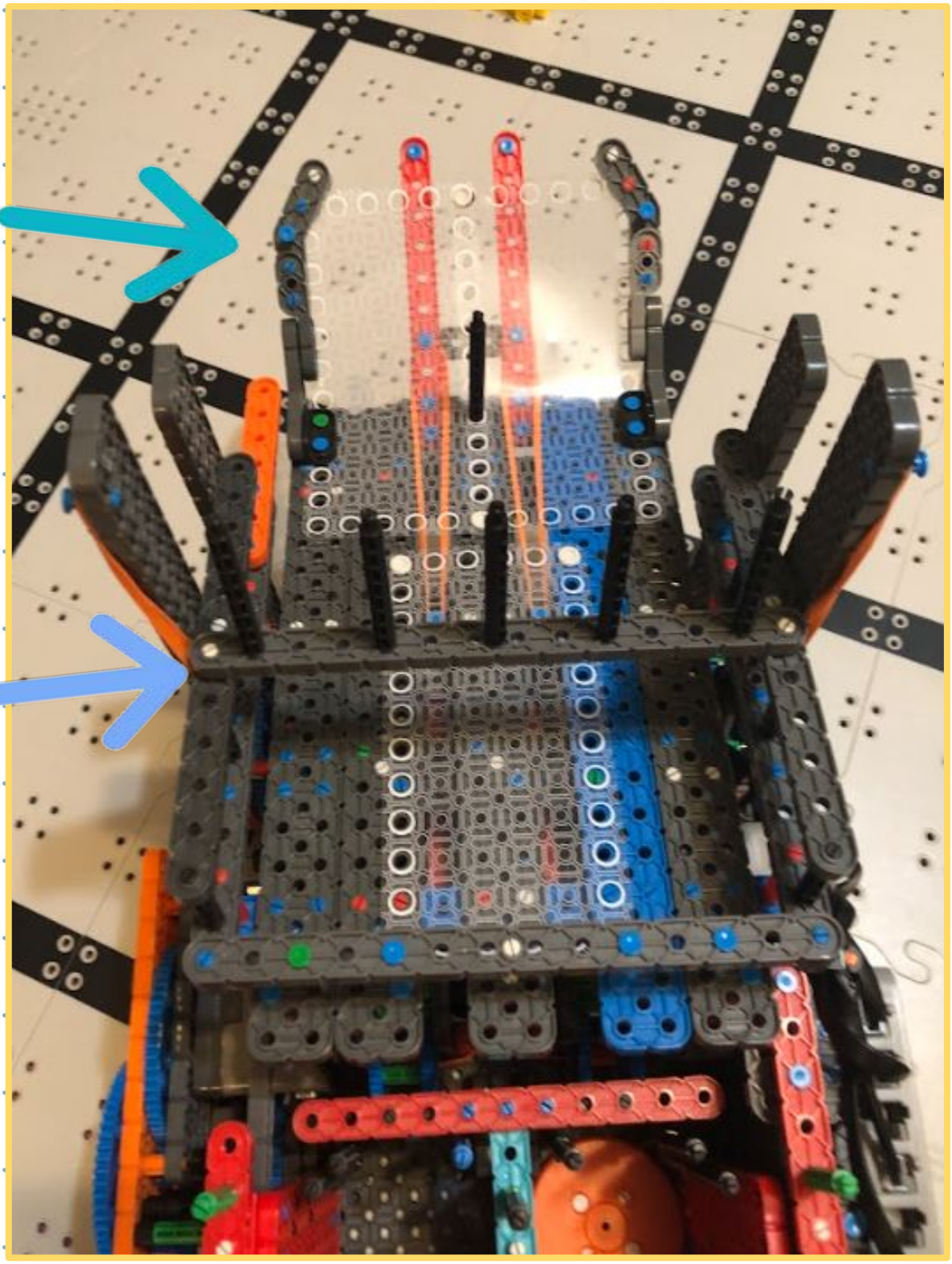
Before, our ramp design was not ideal, because it was flimsy. This design is sturdier than the first ramp we made. The beams are supported with more pieces, so that it can't be moved easily. Previously, the ramp would wobble from side to side, but now the ramp is stationary and more consistent. This ramp will be better than the old one because pieces won't easily fall off and the pucks will not roll off the ramp.

Project Sturdier Ramp Idea

Name kristen Date 11/12

Keeping the Pucks Flat:

when the pucks roll down the ramp, sometimes they would fall into the shooter section vertical. So, we added a stopper bar to keep them horizontal. That way, the pucks would land flat and it would be easier to shoot them.

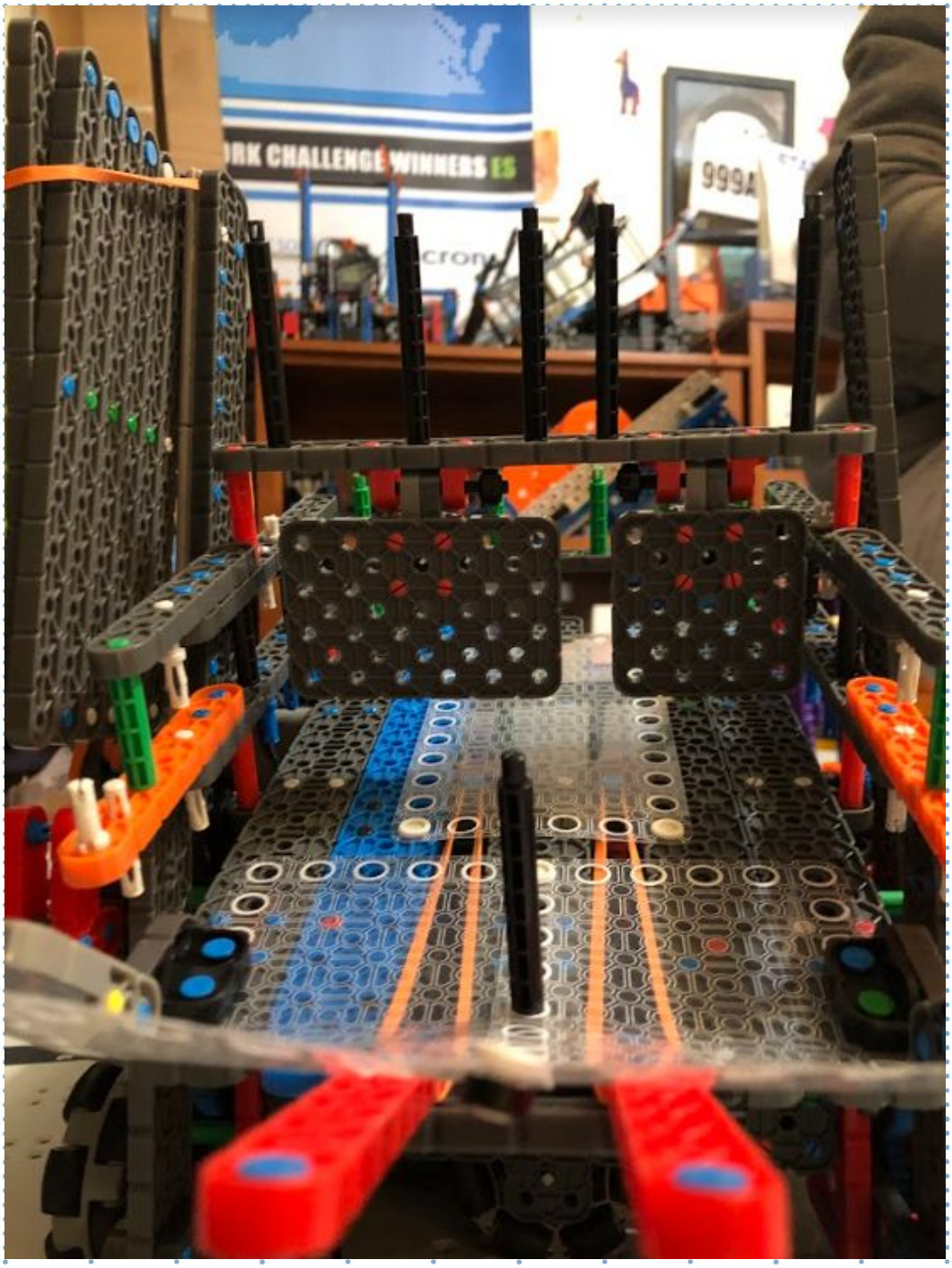


Project Adding a Bar to the Ramp

Name kristen Date 11/12

Some more new additions:

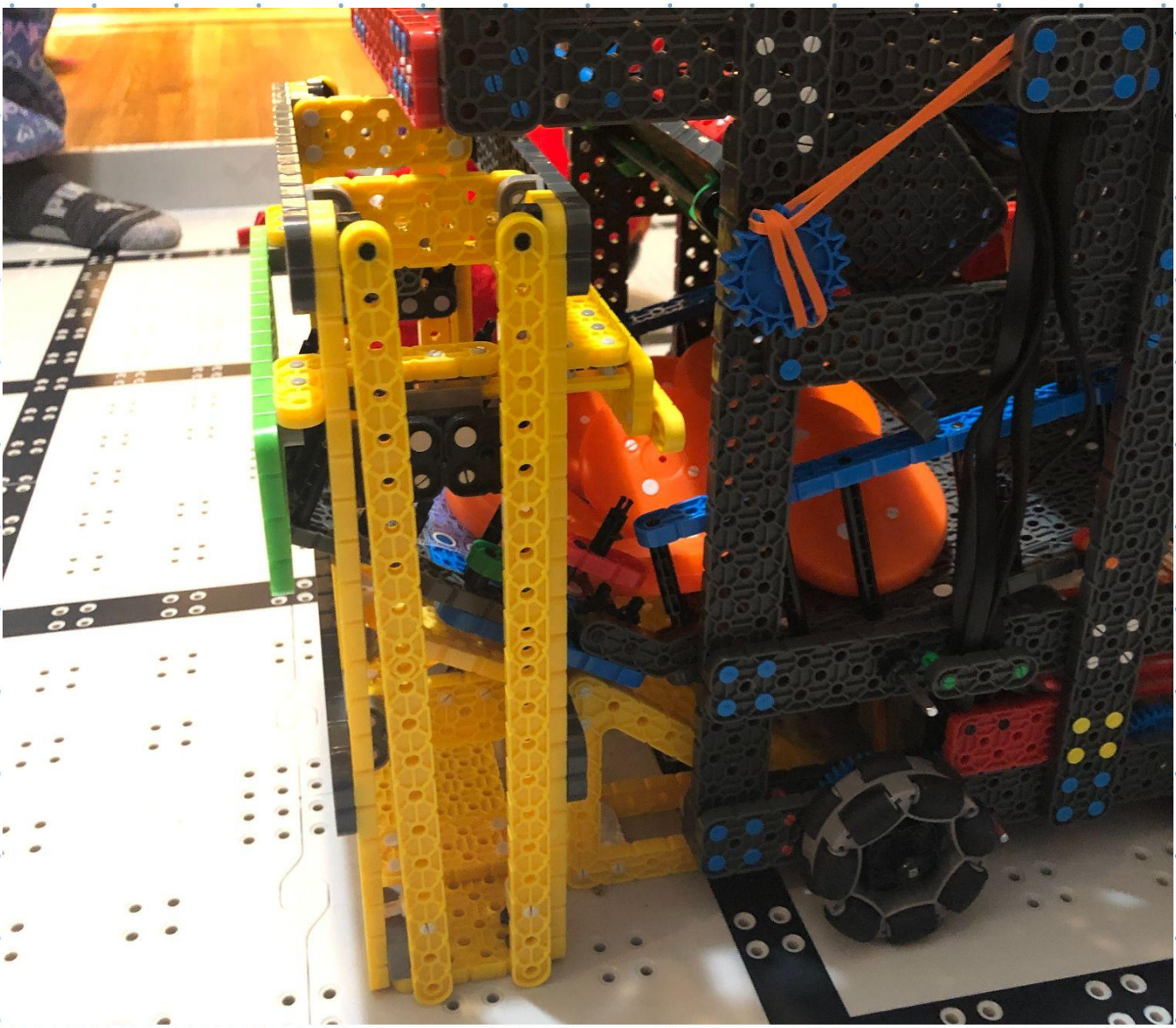
1. Last week we added something to keep the pucks flat. But, we soon learned that that wasn't enough. So, we added another section that works like a flap. This section also slows down the speed of the pucks. When they are loading into the shooter, they don't all rush into it at once.
2. This little addition was to help the ramp hook onto the yellow dispenser. We designed it so that the pucks will roll directly onto the ramp.



Project Additions to the ramp

Name kristen Date 11-27

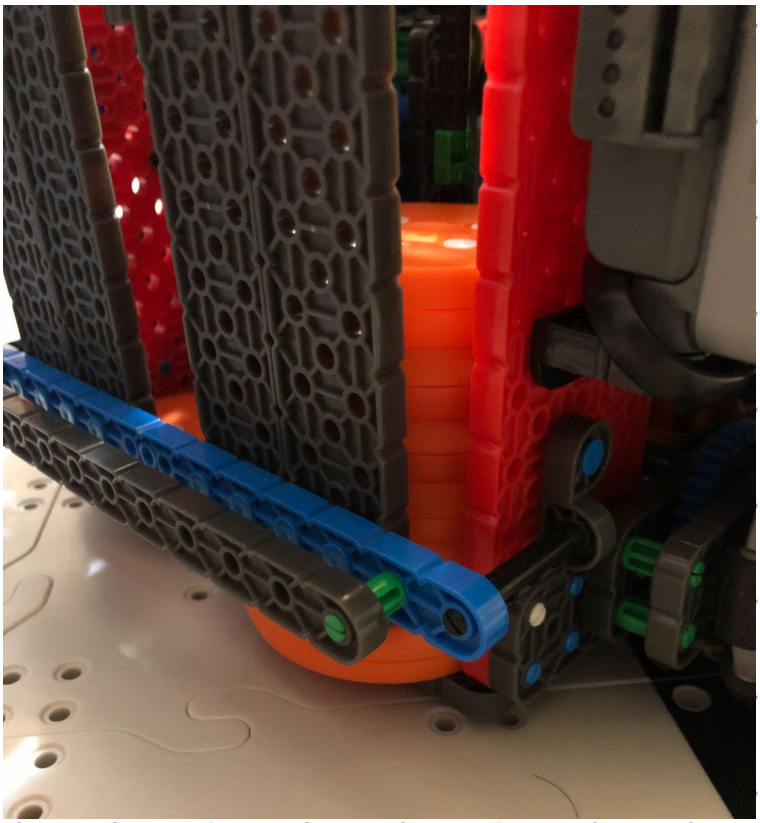
How it works:



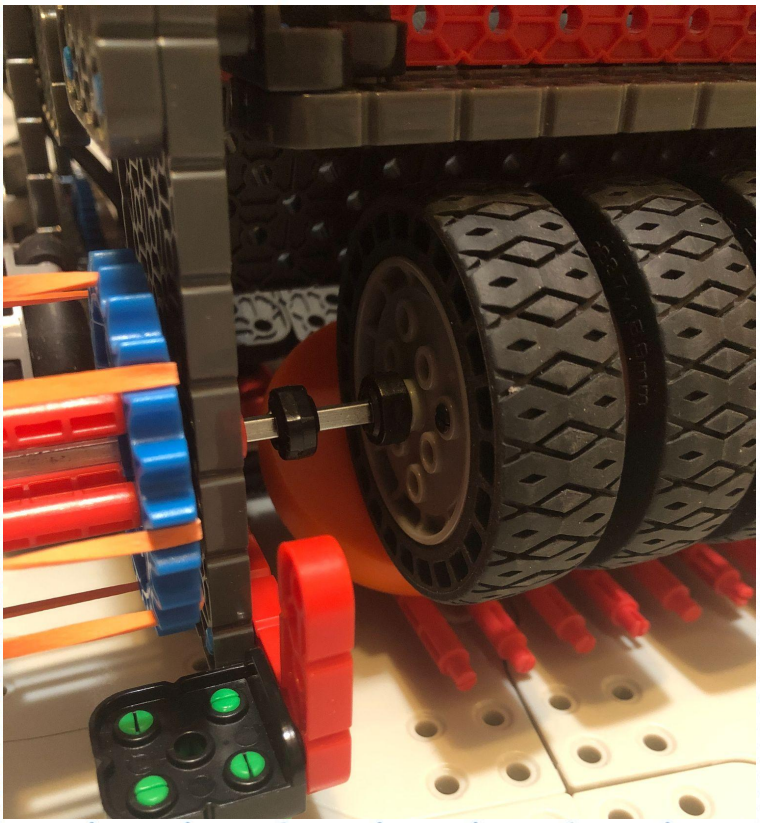
The ramp goes under the yellow dropper, hooks up, and the robot drives back. Then, the ramp goes down, holding all 9 pucks.

Changing the Ramp

BEFORE



AFTER

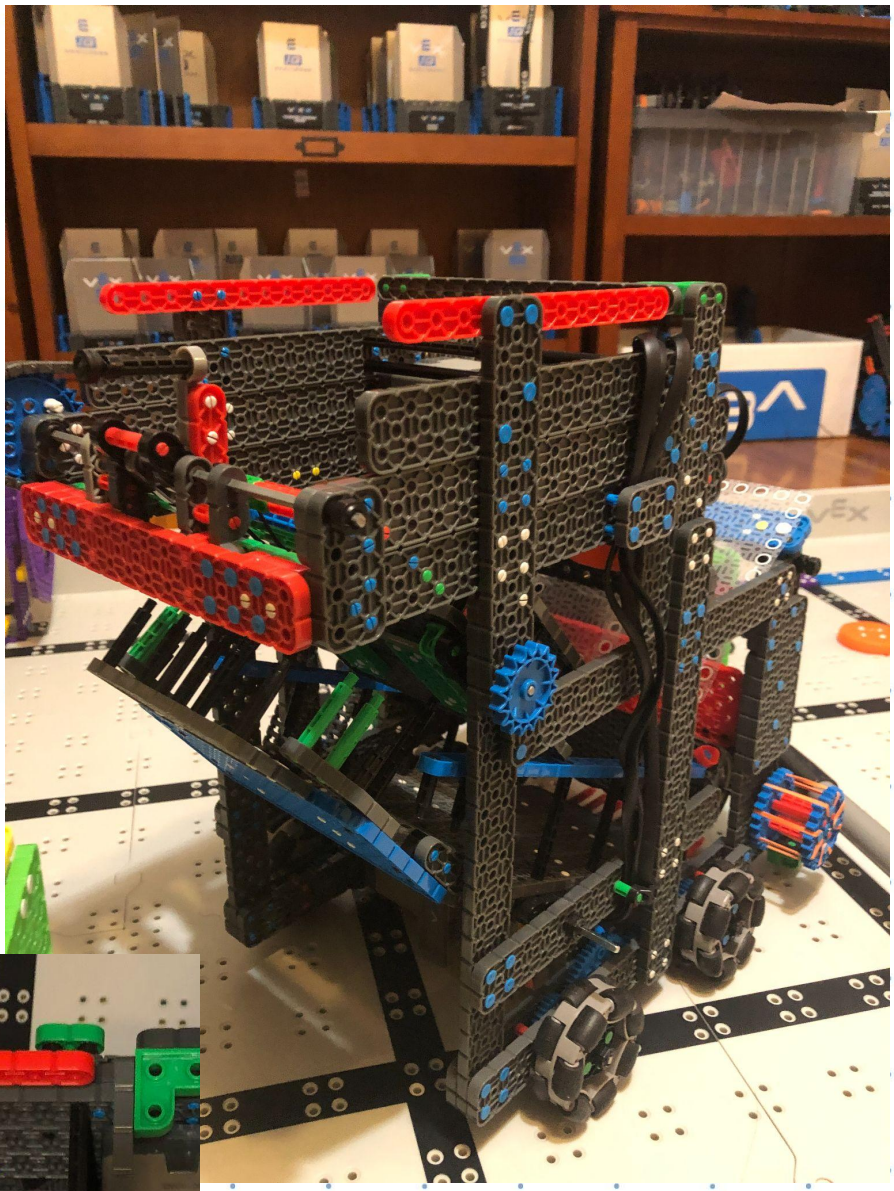


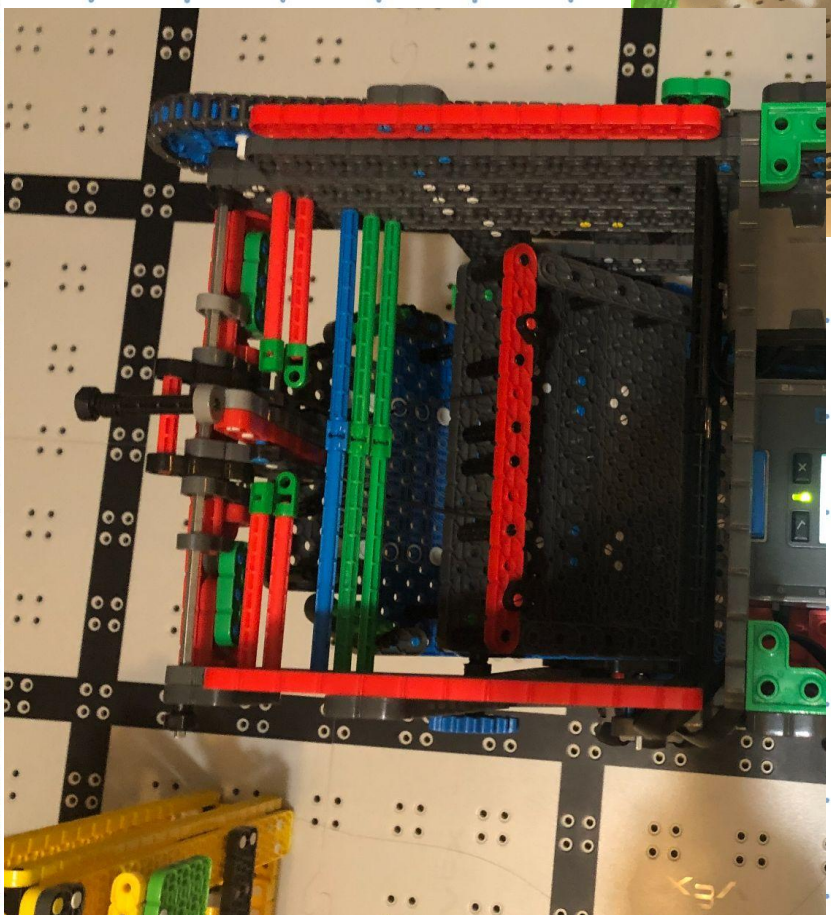
We decided to remove the section where the pucks fall into. Instead of having the pucks stack up into uniform columns, we changed the pucks to go straight into shooting. This way, we wouldn't have too much time spent on trying to load the pucks, and also we wouldn't have the problem of the pucks not loading into the columns in the correct orientation (the pucks can fall into the robot randomly, making it easier to avoid mistakes).

Project Removing the puck holder

Adding a New Layer?!

Why we need another layer: From before, we experienced the pucks getting stuck on the bottom layer because of overcrowding.



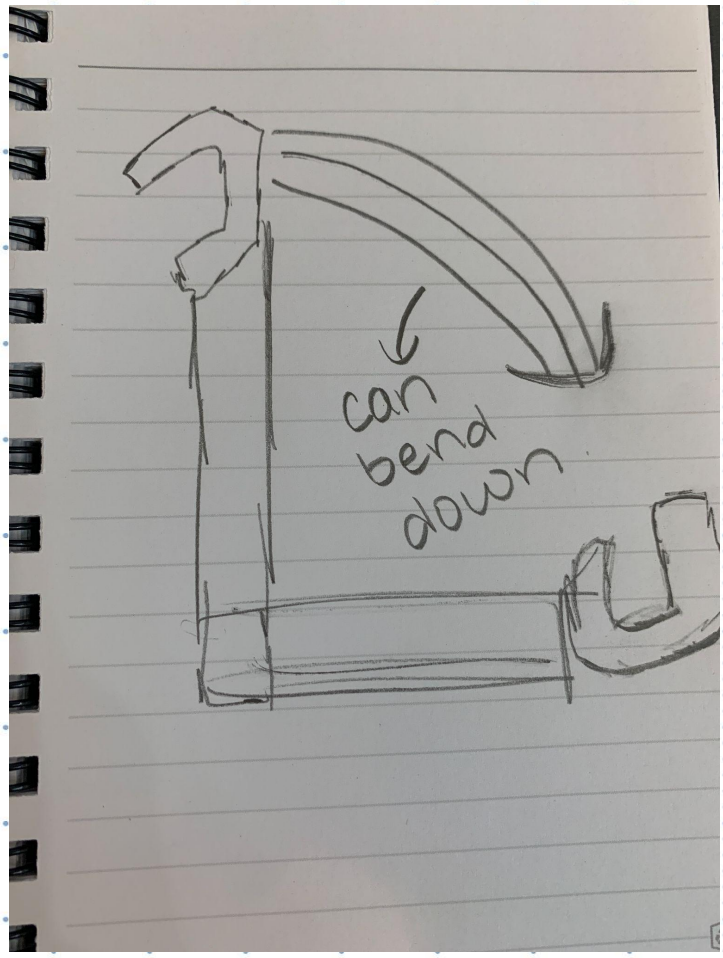


So, we added a second layer for a few of the pucks to sit in, making it less crowding at the bottom (shooter area). While designing, we made sure to take account of the starting height limit.

Dispenser and Touch-down Mechanisms

Hook for Yellow Dropper/Blue dispenser

To the left, is a quick sketch from the last robotics meet describing a hook that can release the yellow and blue dispensers. This hook is effective because it allows us to use one motor for two dispensers.



Problem:

A problem we encountered was that the hook was very weak and not as reliable as we thought, which made us innovate our idea to make it stronger/lightweight at the same time.

Project Yellow/blue dispenser hook

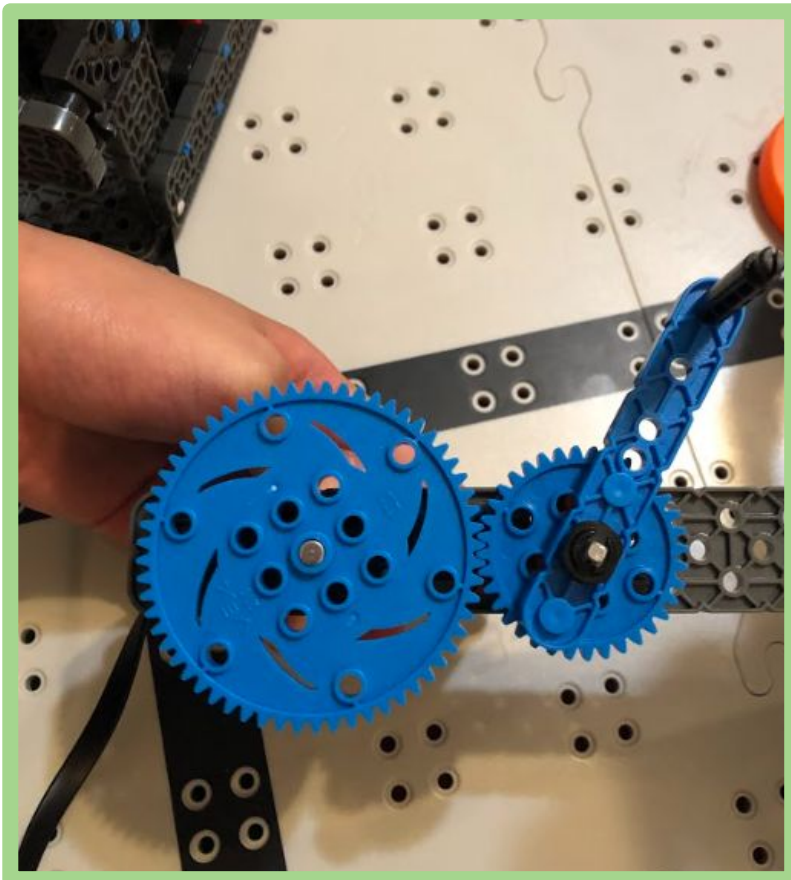
Hook for Yellow Dropper

This hook design is made to drop the pucks from the yellow dropped. we plan to put is near the bucket. The robot drives behind the yellow dropper, then goes forward to hook onto it. The hook will have enough force to pull back the dropper so that pucks can fall into the robot. But the hook's rubber band tension is designs so that it can easily slide out of the dropper's edge.

2nd Hook for Yellow Dropper

Here is another hook we designed for the dropper, but instead of a hook, there is a wheel. We tried using a wheel so that it can easily roll into place. The problem was that since it rolled forward and back, the wheel didn't have the force to pull out the dropper. We tried using new pieces that only turn one way, but it was hard to attach the pieces. Overall, this design doesn't work yet.

Project Designing a hook for yellow dropper



Hook for Purple Dispenser pt 1

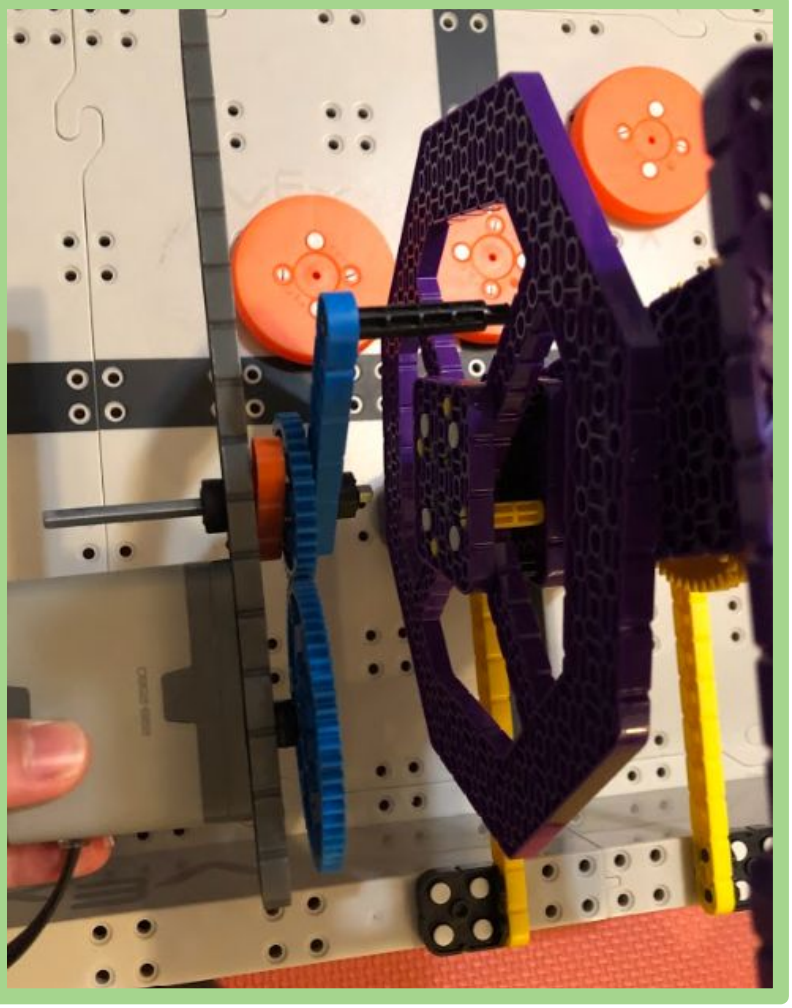
This hook is a standoff attached to a gear. The gear ratio of the mechanism is 3:1; making it faster than without a gear ratio. The standoff is placed in between the holes of the dispenser, and spins to allow the pucks to fall.

Problems:

- Might be hard to aim into the holes.
- Since it's on the side of the dispenser, the pucks might not fall into the robot's bucket.

Ideas for Improvement:

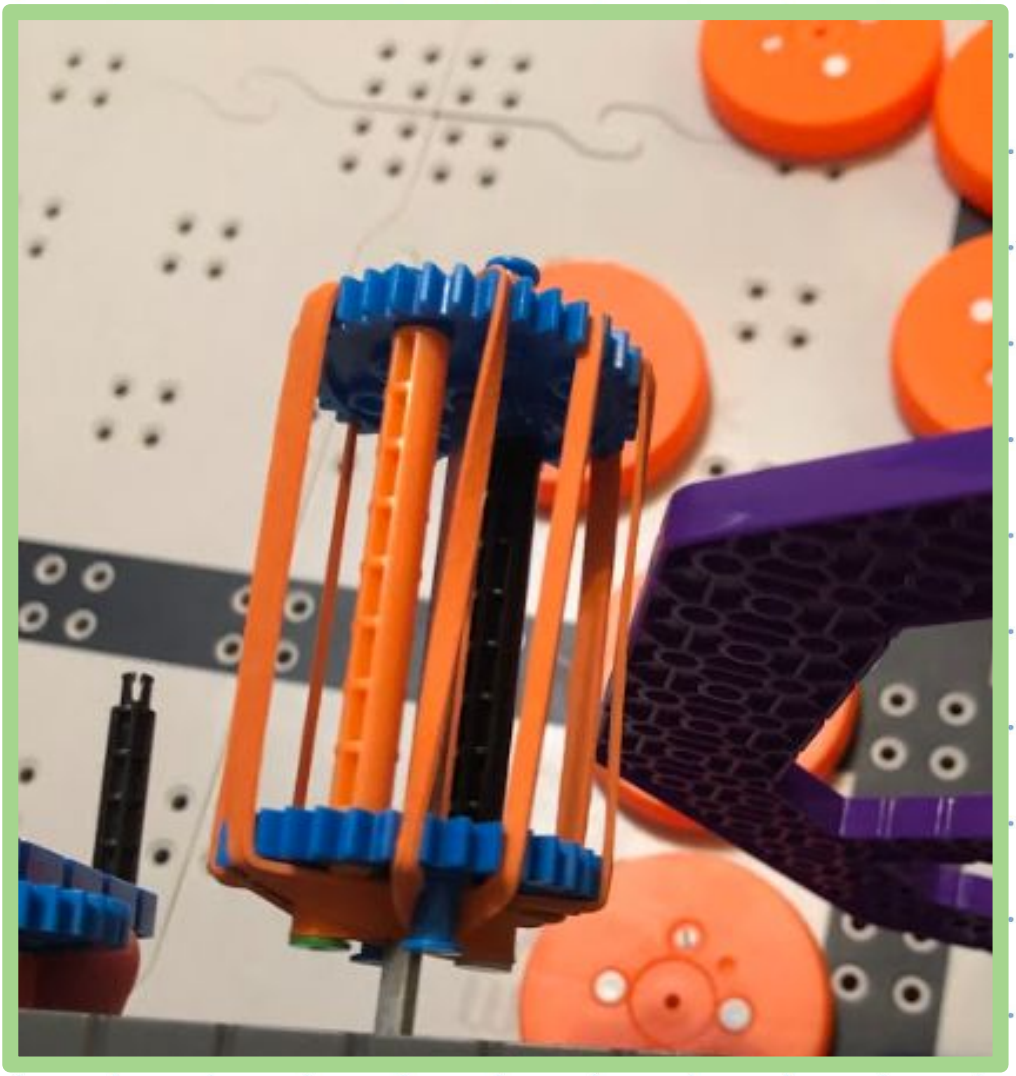
- Somehow making it be on both sides of the robot only using one motor to be easier for the drivers



Project Hook for Purple Dispenser

Name kristen Date 9/23 Page 41

Hook for Purple Dispenser pt 2



The Idea:

- using our knowledge from last year, we designed a rubber band roller that turns from a motor. The grip from the rubber bands will turn the dispenser.

Problems:

- was a very slow design. We couldn't gear it up because the rubber bands on the gear were getting the gears stuck, and making it slower.
- May be hard to attach to robot.

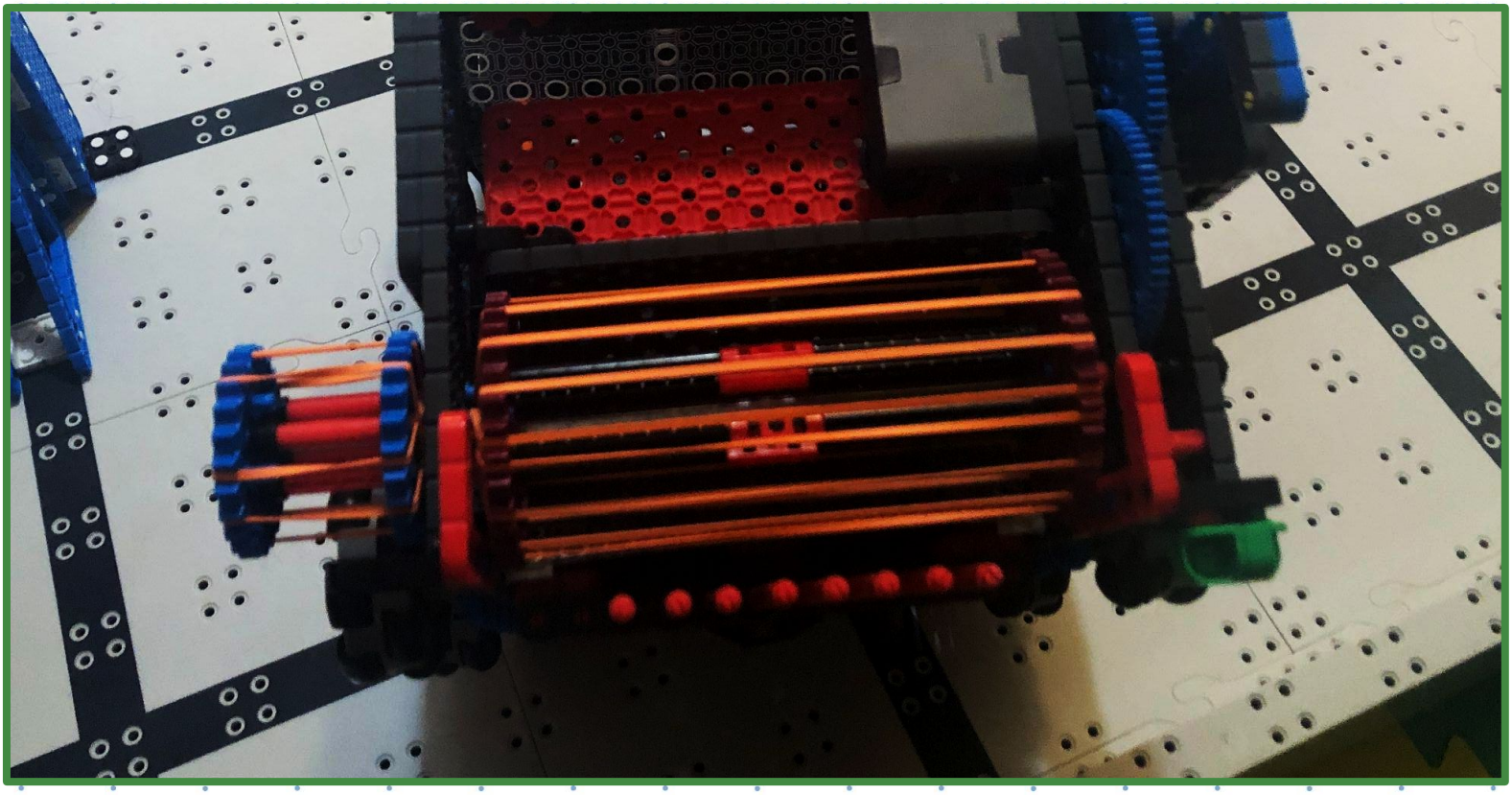
Hook for Purple Dispenser pt 3

The Addition:

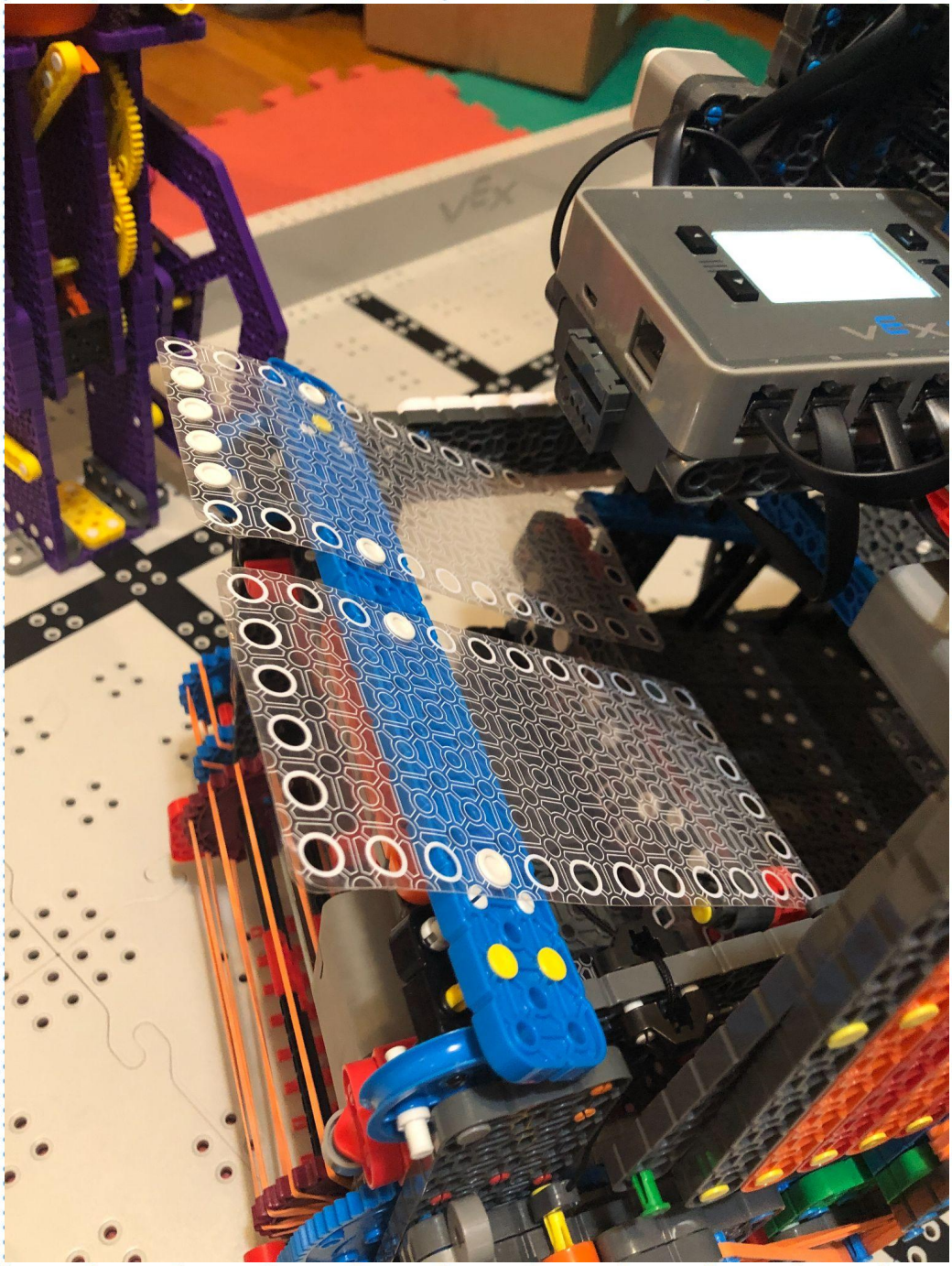
- We added the collector design next to the shooter, so that we could spare a motor. By putting the two mechanisms together, they could be on the same axle and we could use the extra motor for something else, like the shift wheel

Problems:

- Sometimes, it was difficult to line up the roller with the purple tower.
- The roller sometimes didn't go into the robot's basket.



Hook for Purple Dispenser pt 4



Changes:

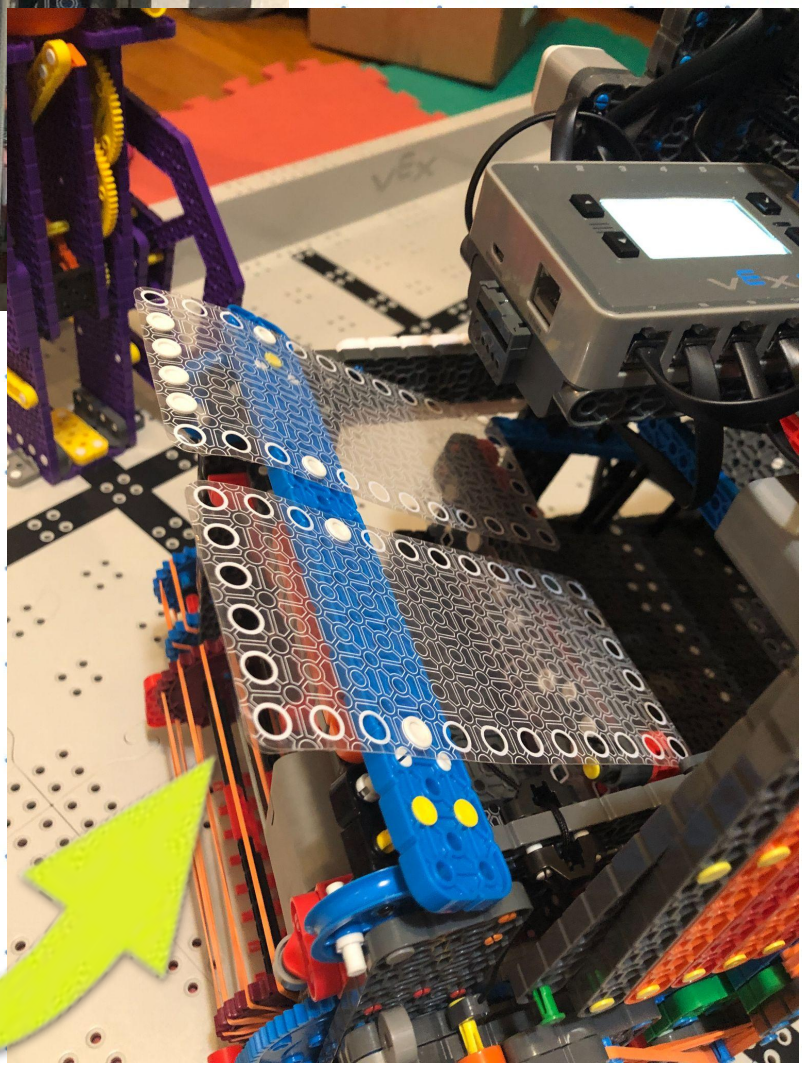
- A change we made to the purple dispenser is making the shooter a collector too. We realized that since the two mechanisms are connected, why not make the shooter a collector too?
- This helped us by making it easier to aim for the purple dispenser.

See more

Project Hook for Purple Dispenser

Hook for Purple Dispenser pt 4.5

:Before



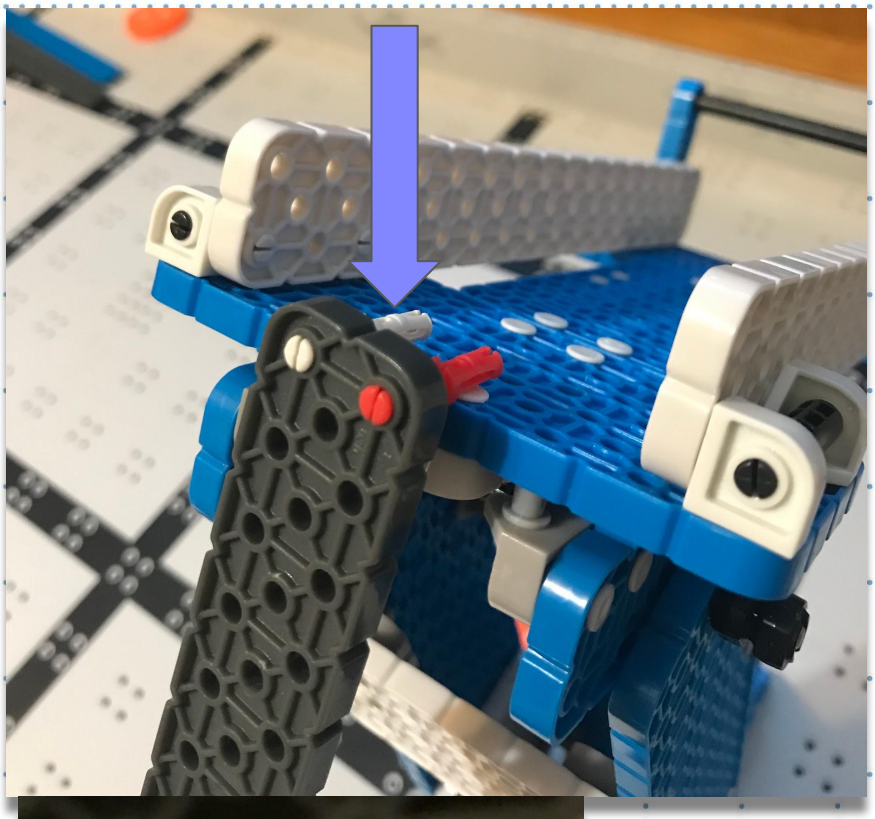
After:

Plastic sheets are farther apart in the new version so that is covers more surface area. This helps by making it so that the pucks have more space to land into the robot.

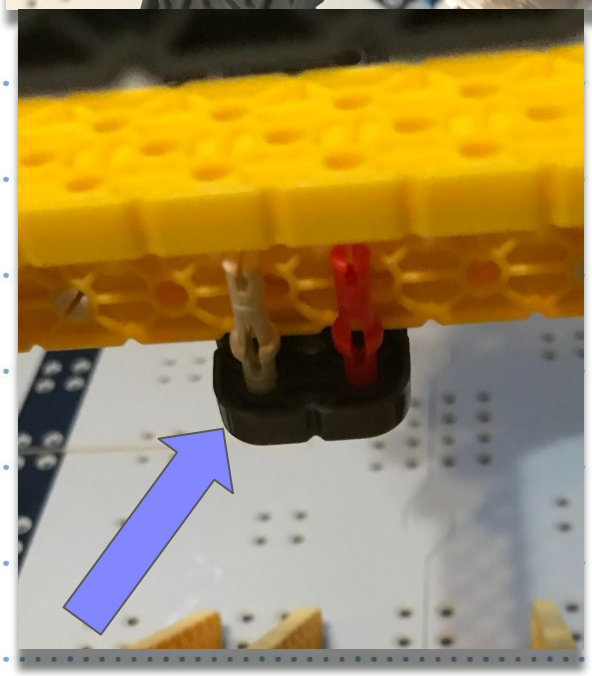
Project Hook for Purple Dispenser

Name kristen Date January 2023 Page 45

The Double Hook



The idea for this hook is very simple. In the pictures to the left, there is a beam with pegs sticking out of the end.

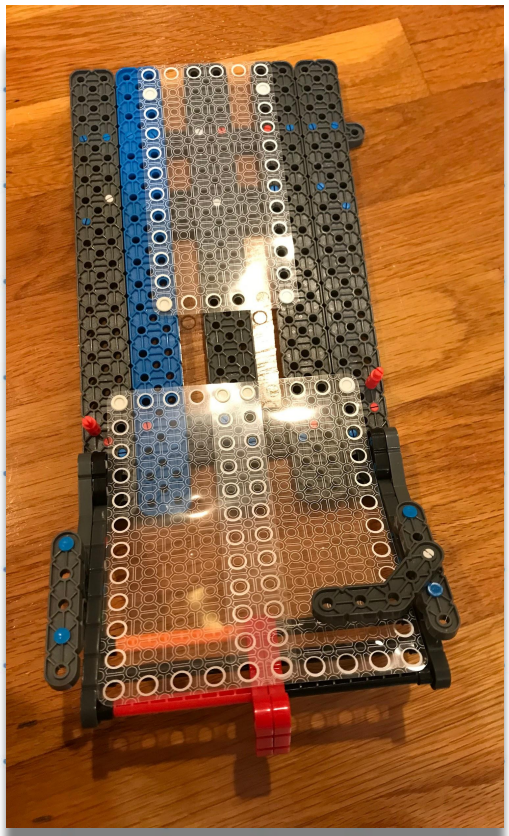


The way the design works is the hook pushes down for the blue dispenser. On the yellow dispenser, it goes under, moves up, and then the robot drives back. By driving back, the yellow dispenser will release the disks.

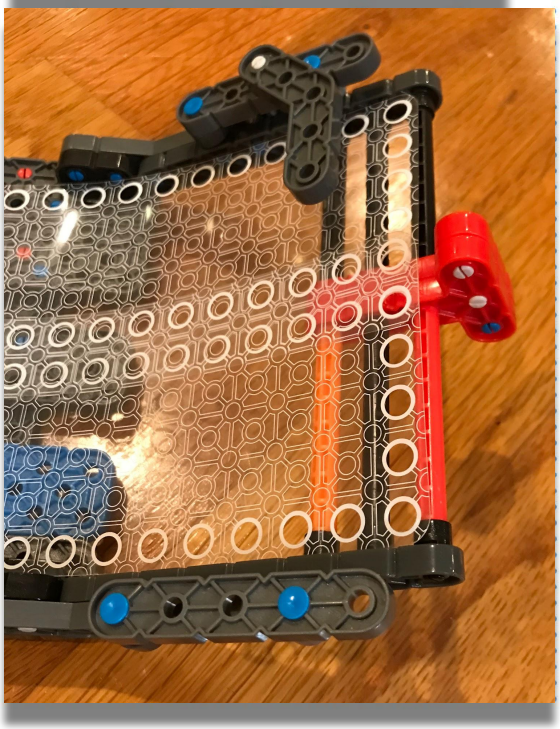
Project Hook for Yellow and Blue Dispenser

Name Sammy Date 10/7/22

The Double Hook 2.0



This design works just like the previous double hook. THe only difference is that it is attached to the disk ramp and it is hooked on both sides. This allows the hook to hook down onto the blue disk dispenser and also hook the other way onto the yellow disk dispenser.



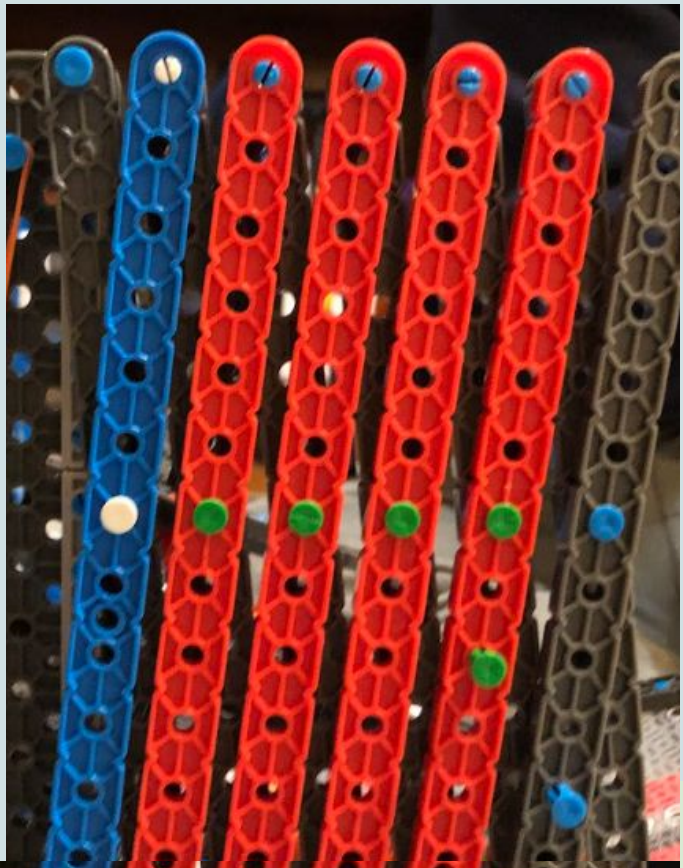
Pros

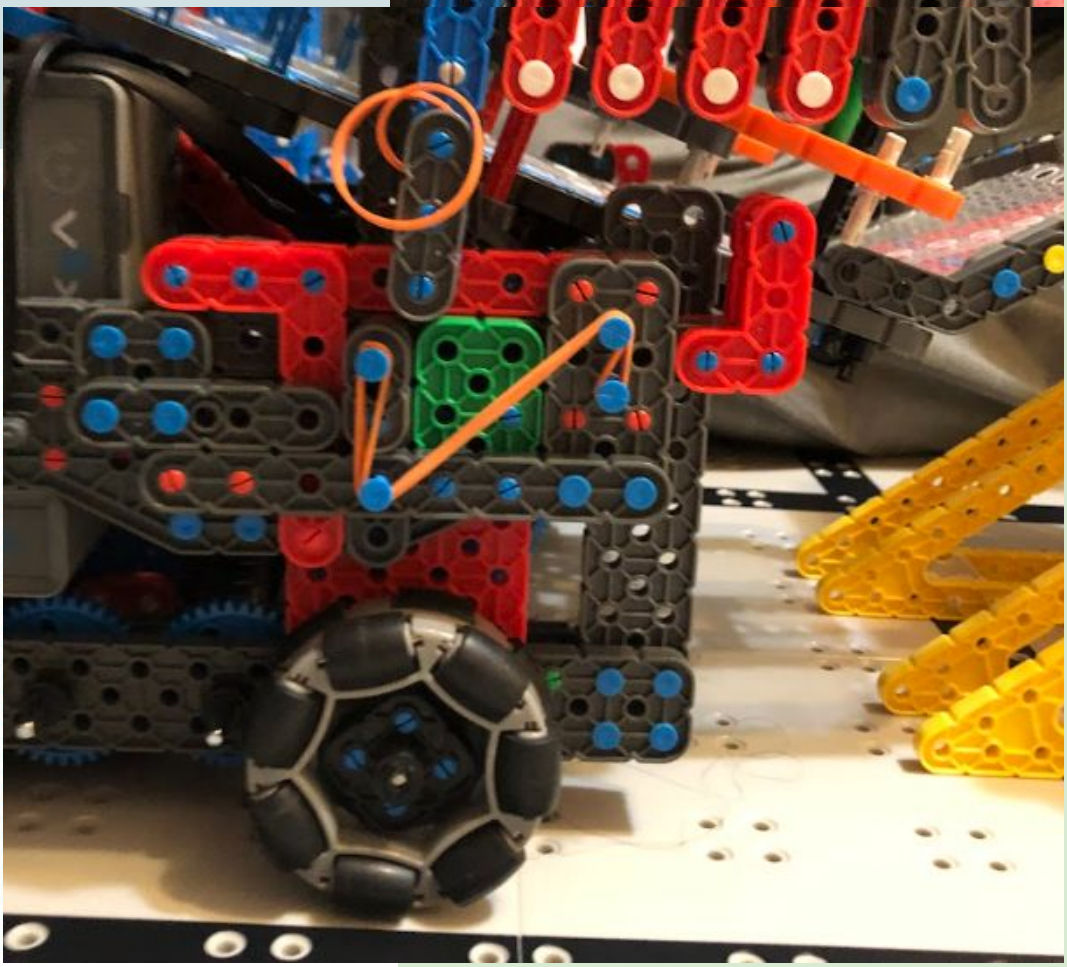
This design is extremely simple and effective. It also takes up very little space and can be easily incorporated into the robot.

Project Hook for Yellow and Blue Dispenser 2.0

Name Sammy Date 10/14/2022

Working on the Touchdown





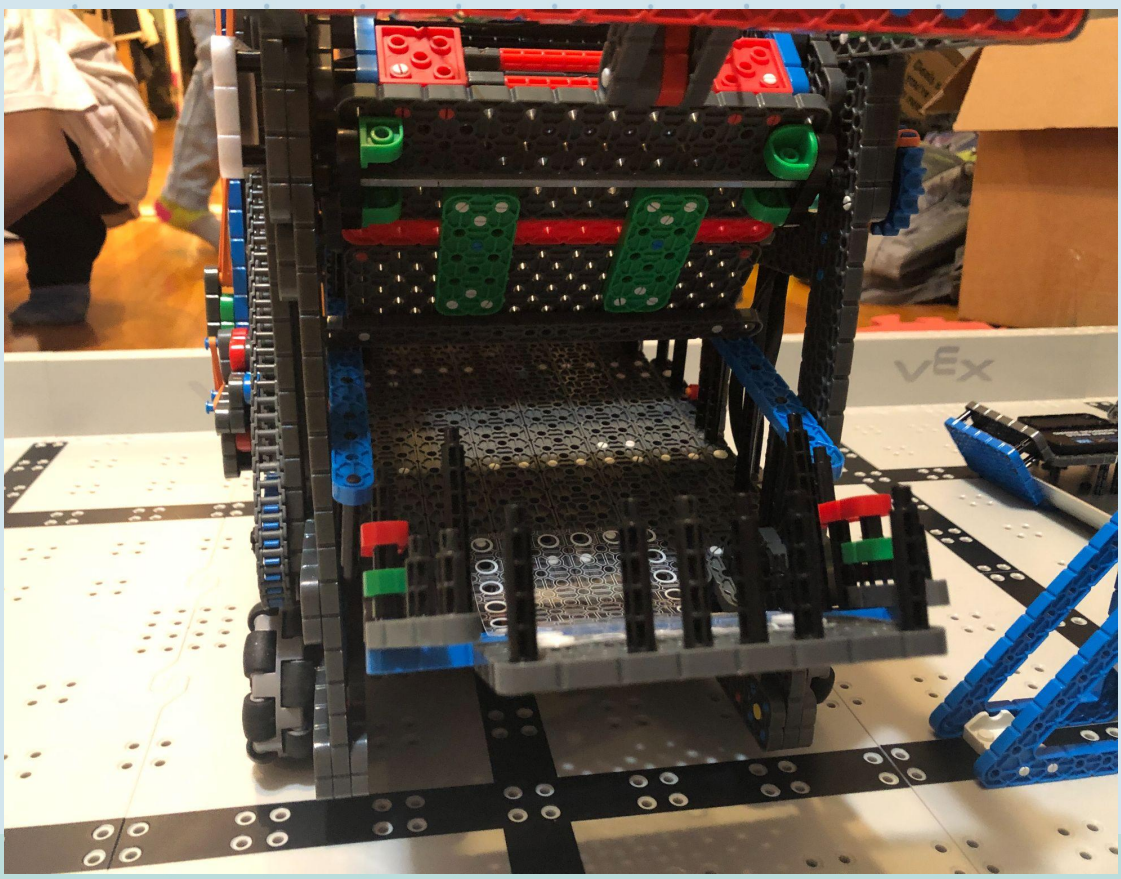
When the small red "L" piece comes into contact with the wall of the divider, the mechanism will shoot out onto the green section of the board. This mechanism will only be used at the end of each run. During the designing process of this mechanism, we realized that it wouldn't be very good if something were to accidentally hit the "L" piece, and release the long arm. So, we added extra locks (kind of like baby locks) that make sure they the arm goes out at the wrong time.

More info: Next Slide

Project Working on the Touchdown Mechanism

Name kristen Date 12/31

Working on the Touchdown



This is how the lock works:

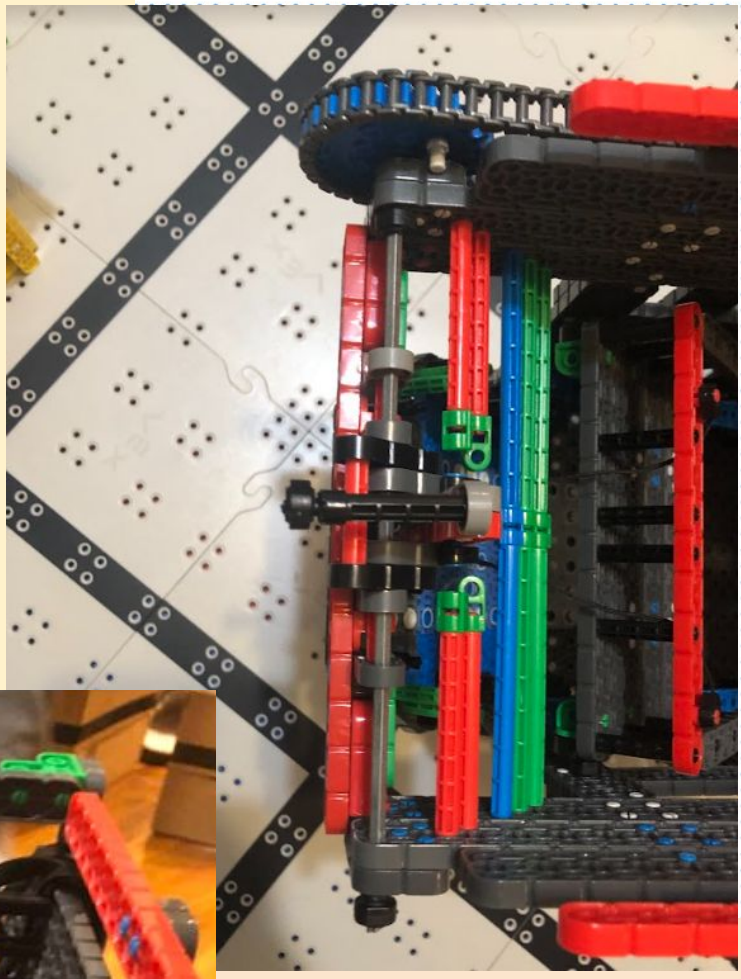
1. Make sure the ramp is all the way up. The mechanism will only release if the ramp is up. This was a way we added a "baby lock". We didn't want to accidentally activate the touchdown, so we added this.
2. Hit "L" piece
3. Moves pieces that are connected to it out of the way
4. Leaves a gap the releases the arm
5. Arm goes outwith the tension of the rubber band.
6. Touch the board
7. Drive robot to adjust where the robot lands
8. (arm is not retractable)

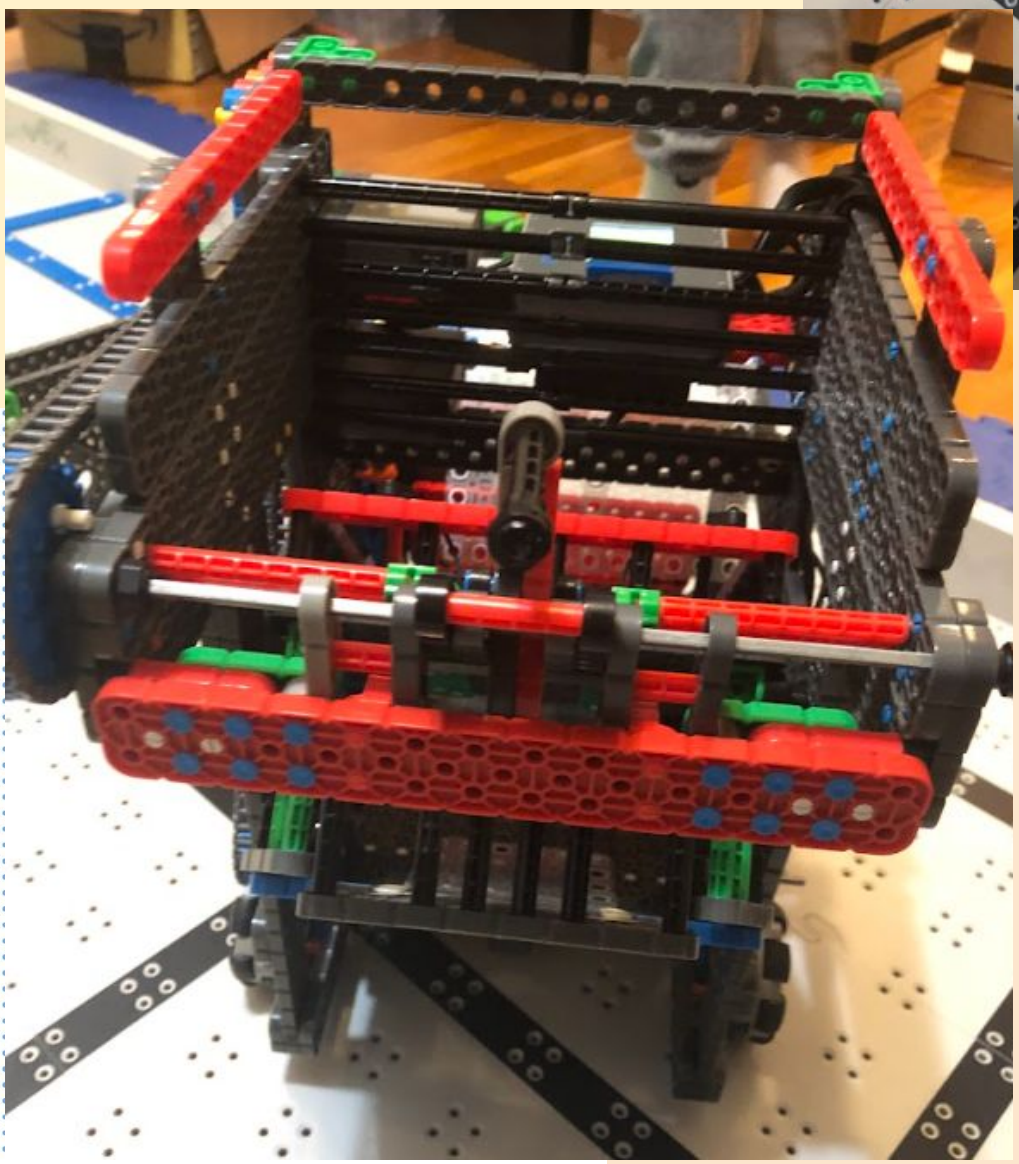
Project Touchdown

Name kristen Date 1/4

A Better Hook Design!

This hook design is much lighter than our previous one. Last time, we encountered the problem of the hook not be sturdy enough—it would sometimes be hard to control.



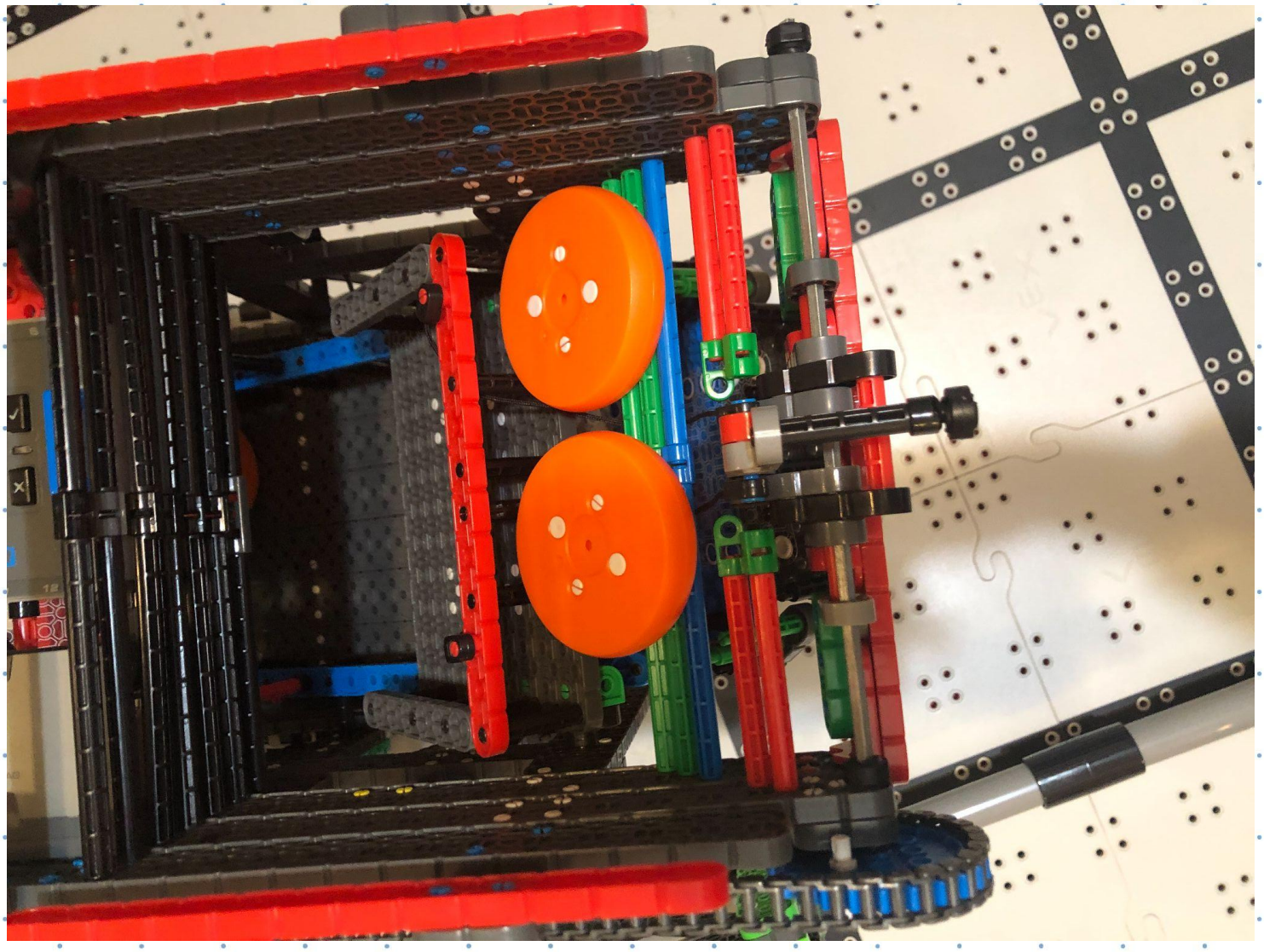


So, we made this design to save space, and make it more reliable. Also, it was easier to access the pucks on the blue dispensers due to the angle it and direction it moves.

Project Better Hook Design

Name kristen Date 12/31 Page 50

Hook Design Issues



we encountered the problem of having the pucks fall into the wrong place. Instead of going onto the ramp, it gets stuck above the standoffs, causing it to get stuck. This problem only occurred while driving. After a few times we noticed that this occurred because we left the bottom layer ramp up in shooting position, letting the ramp have space to go down.

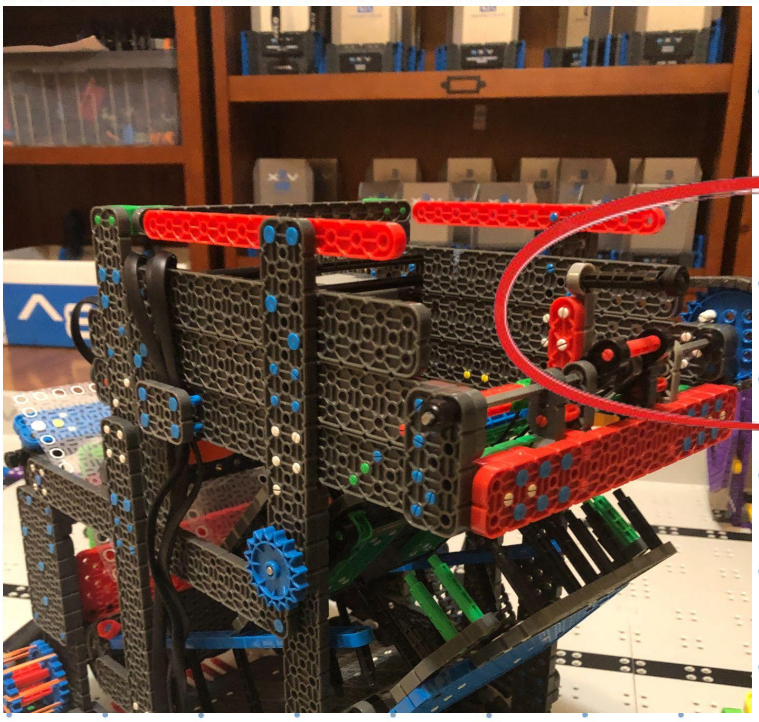
Project Problem with hook design

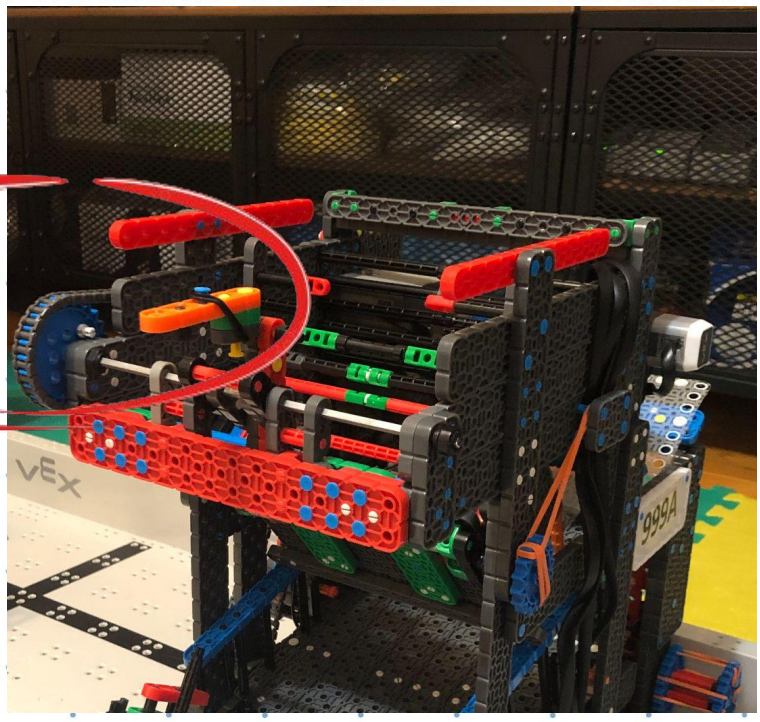
Name kristen Date 12/14 Page 51

Continuing to improve!

After testing out the standoff hook design, we noticed that it would break off easily. So, we worked on a sturdier design that uses beams instead of standoffs. Standoffs are weaker than beams because they are much thinner. We also added a black band around the beam to keep it intact; without it, the beam would fall off sometimes. We have not noticed any issues with the new blue hooker yet!

Before ..... After!





Project Improving the blue hook

Name kristen Date 1/14

Strategy

Second Generation!

This year, we decided to use the new second generation pieces to build our robot. When looking through them, we saw many familiar pieces, but also an abundance of new ones. We are excited to test out the new pieces and figure out how they work. Also, the new brain, controller, and batteries were really cool too. We were also pleasantly surprised to hear about the new coding platform we will be using this year. In the past few years, our team has been using Robotc, but this year, we will use a variation of the c++ language on VEXcode.

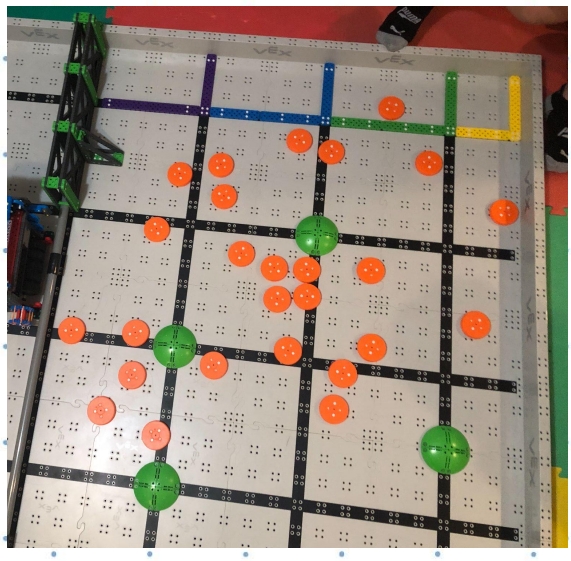
update: While driving our new robot, we noticed that the new batteries last much longer than the old ones. We don't have to charge them as often!

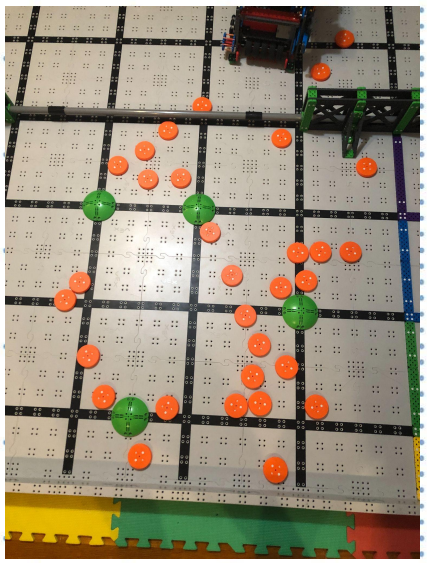
Project Trying out gen 2 pieces

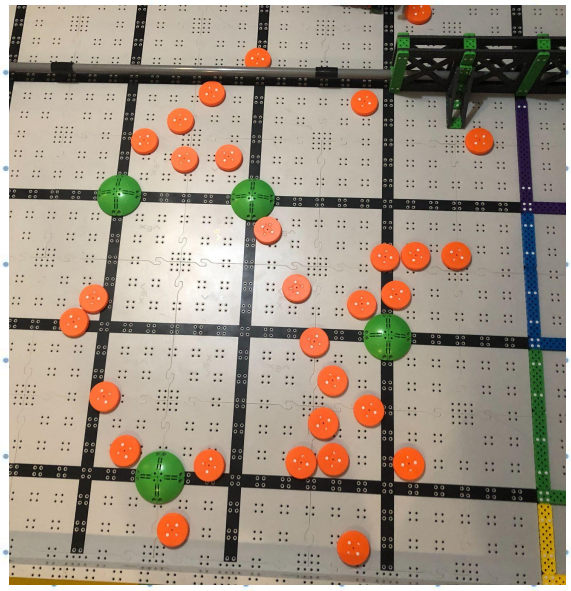
Name kristen Date 12/31

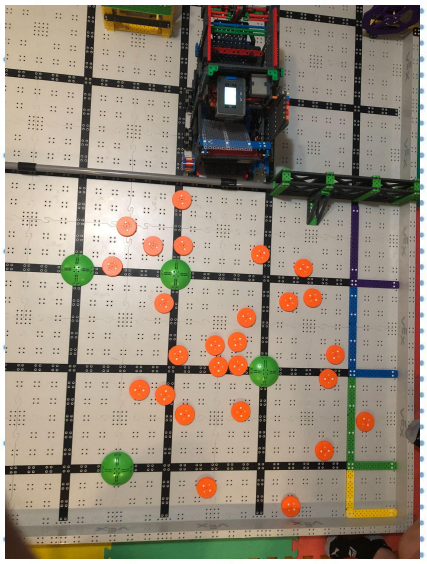
After building the prototype of the robot, we started testing it out. Here are the results we got after our first few test runs!

As you can probably tell, the outcomes of the runs were never consistent. Sometimes we would hit the green stoppers on the board, making the pucks super closed together, while other times the pucks were super spread apart. This could be because of the lack of practice, so we plan to keep working hard. Our goal is to have most of the pucks in the green/4-point section.







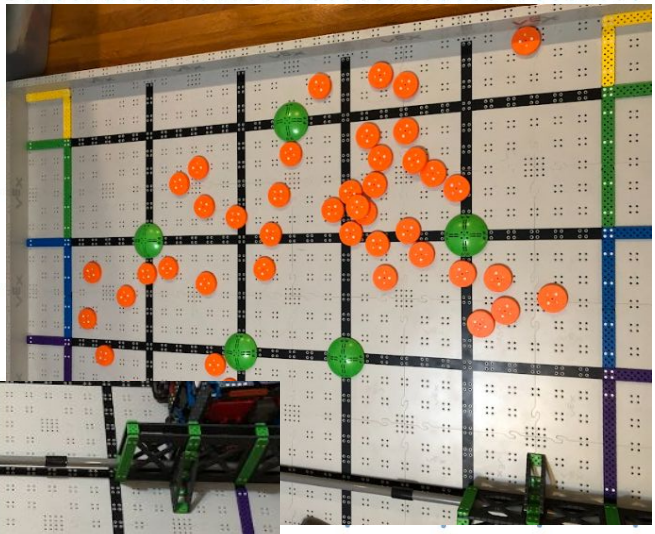


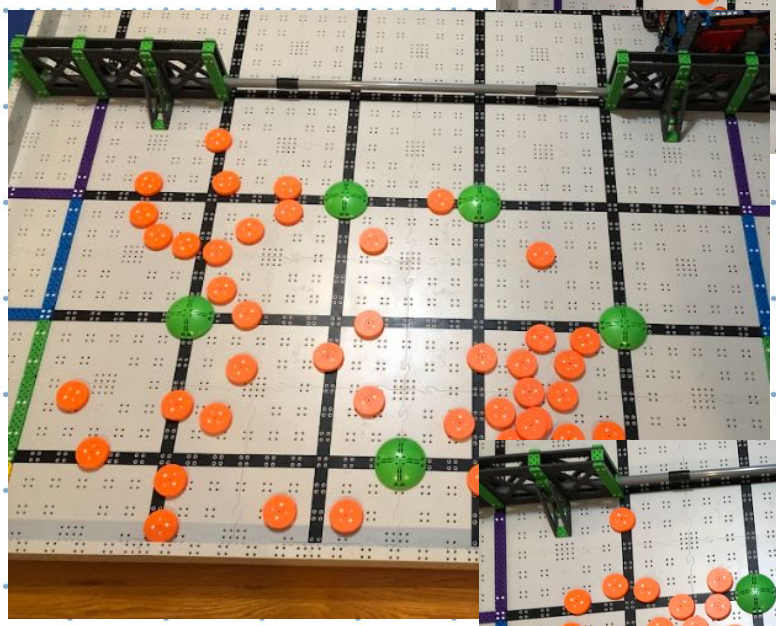
Project First tests runs

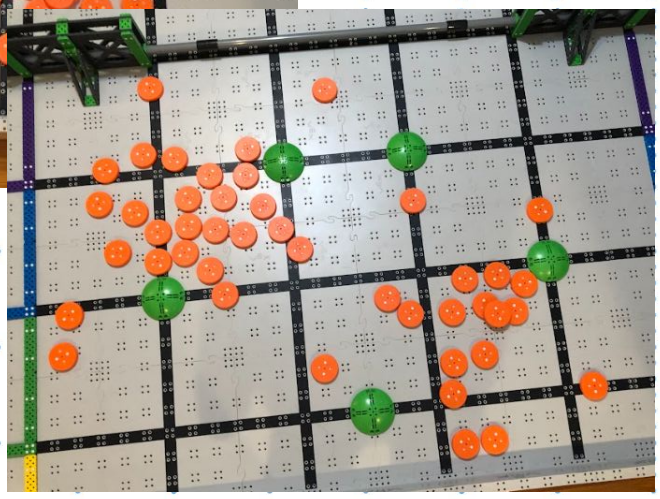
Name kristen Date 12/16 Page 55

We did more runs after changing the shooter design from wheels to rubber bands.

Here are some more driving practice results: After changing the shooter design, we started getting more consistent results. Plus, pucks were less likely to shooter over the fence. This could have also been because we started getting used to the feeling of the robot and shooter.





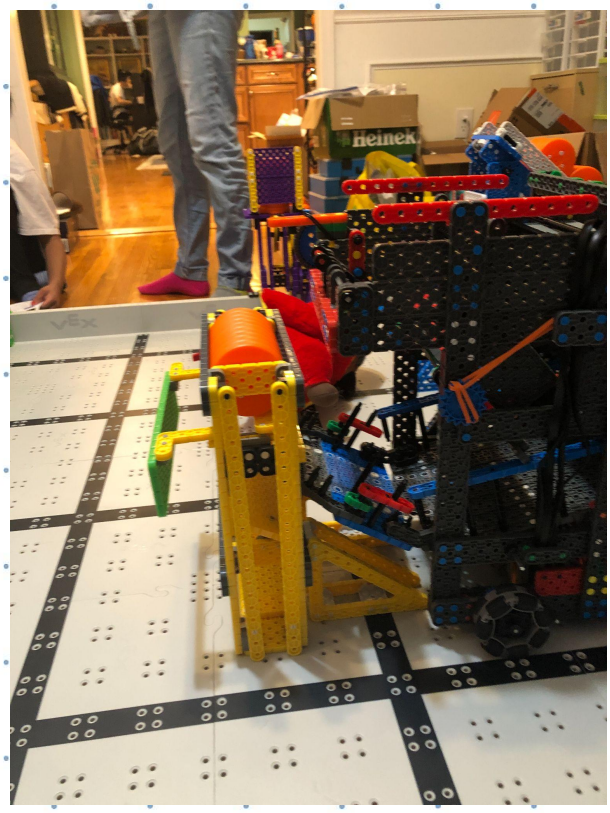


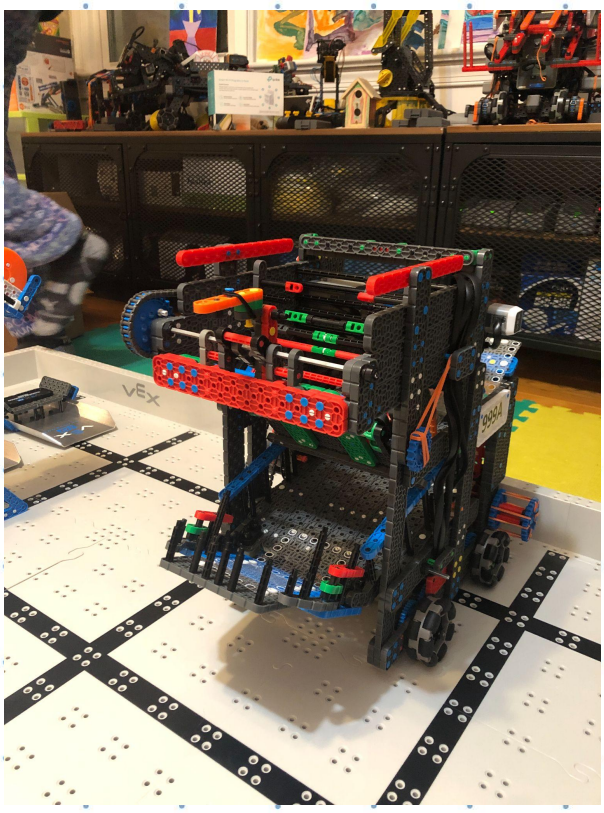
Project More driving runs

Name kristen Date 1/20 Page 56

Starting Position:

We chose this starting position because it was right in front of the yellow dispenser. Being in front of the yellow dispenser would help us during driving and programming. This is because the first part of our puck-collection-route is to grab the yellow pucks. Meaning we would just need to drive forward.





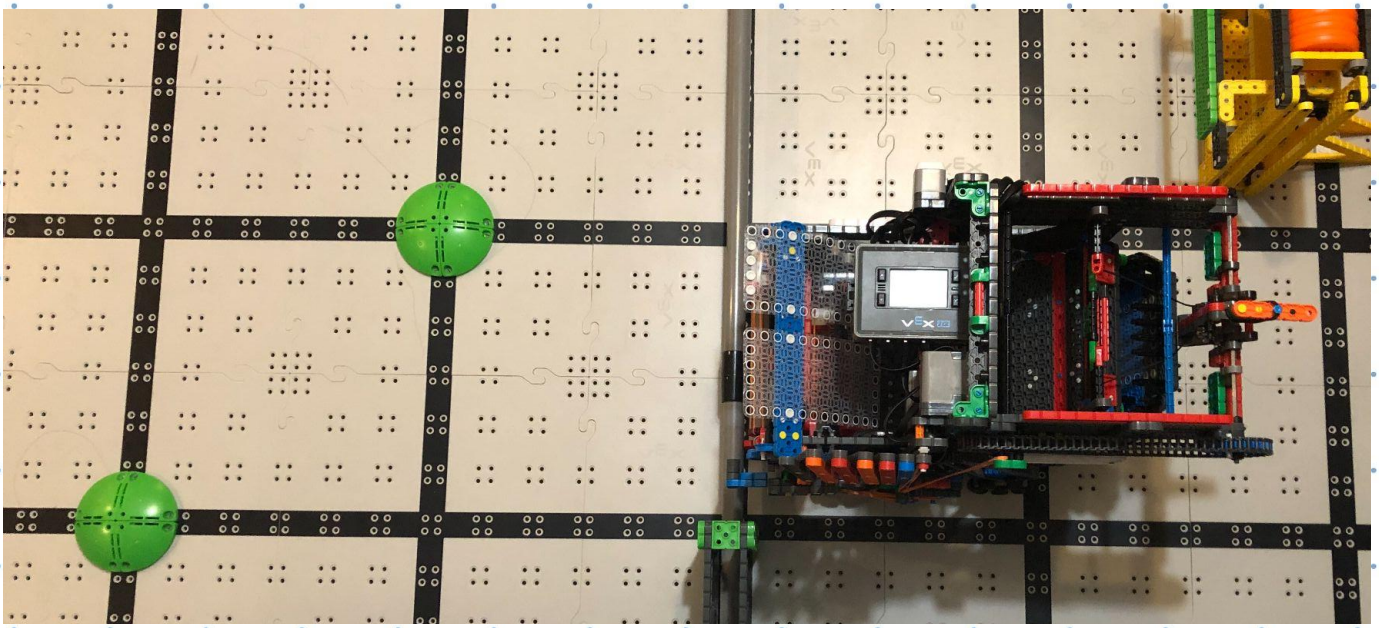
Project Choosing a starting position

Name kristen Date Page 57

Shooting Position:

There are two shooting positions we use, one left of the middle and one right of the middle. They positions are ideal because they avoid hitting the green bumps on the shooting area. If we hit the bumps, the pucks go fly onto an unexpected area, so we try to avoid that. Also, these positions are the easiest for our robot to reach; they are also the fastest.

Something else we make sure to do while shooting is lining up the robot all the way against the shooting bar. This way the pucks don't accidently shoot above the bar (shooting above the bar will result in disqualification, which is something we want to avoid).

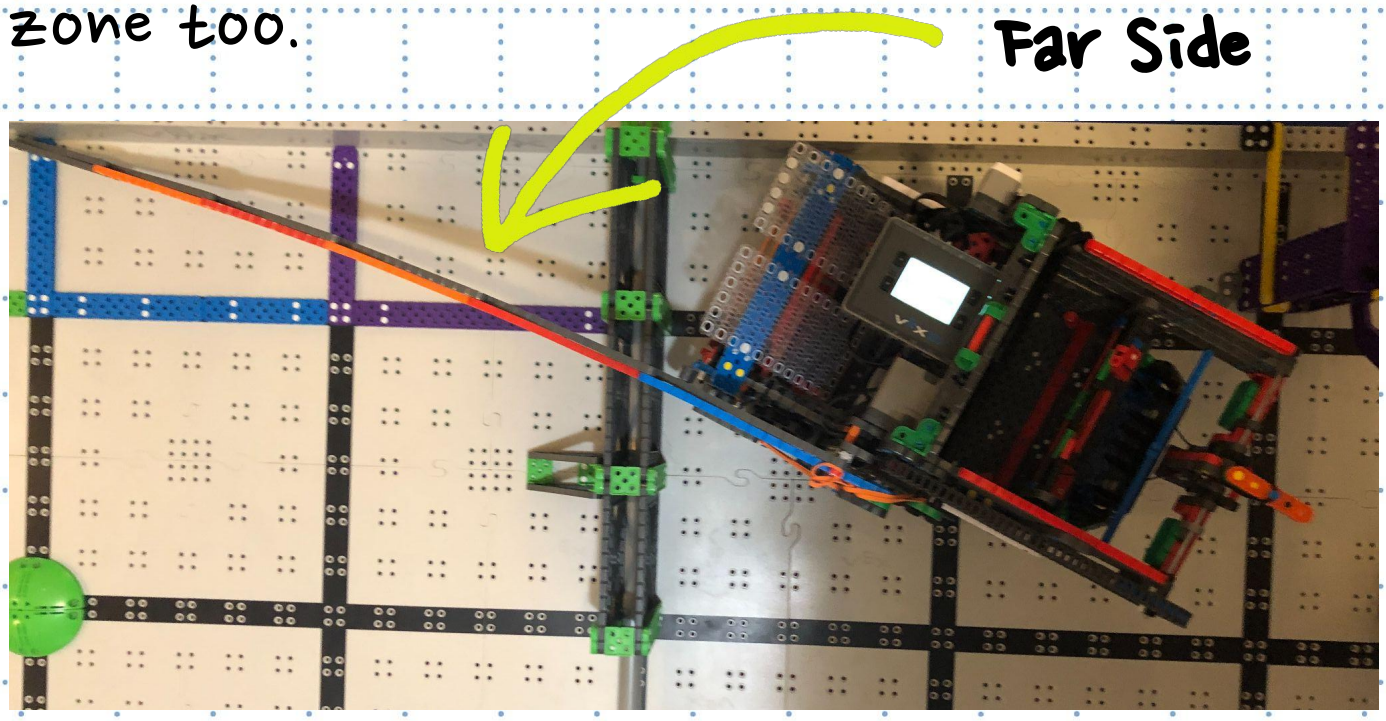


Project Choosing a shooting position

Name kristen Date Page 58

Touchdown Positions:

Since our robot only has one "touchdown" mechanism, it is easier to activate it on the far side of the board than it is on the near side. This is because on the far side, the robot can just shift over, hit the wall, and activate. But on the near side, it has to adjust to make sure the robot in the scoring zone, move diagonally, and make sure the touchdown makes it in the scoring zone too.



Project Choosing a shooting position

Preparing for Our First Competition of the Season

Things we did:

- Assign jobs to everyone
  - Drivers: Sammy, Kristen
  - Programmers: Zoe, Vaishnavi, Vaishali
  - Liaison: Zoe, Vaishnavi
  - Scheduler: Vaishali
- Practice driving
- Finalize programs
  - Teleops
  - Autonomous
- Meet to wish everyone good luck and go to bed early!

Project Preparing for our first competition

Name Kristen Date January 2023

Competition Reflection:

Successes:

- working hard during every teamwork match. Sometimes our runs didn't go as planned, but we pushed through to better next time!
- Due to changing the robot's shooter to using rubber bands, the shooter never got stuck during the competition and was functioning consistently.
- our team worked together well; everyone was on the same page during the competition.
- our team also worked well with other teams. We made sure to meet with each team we were alliances with to make sure to practice a strategy with them.

Project Successes at our first competition

Name kristen Date January 2023 Page 61

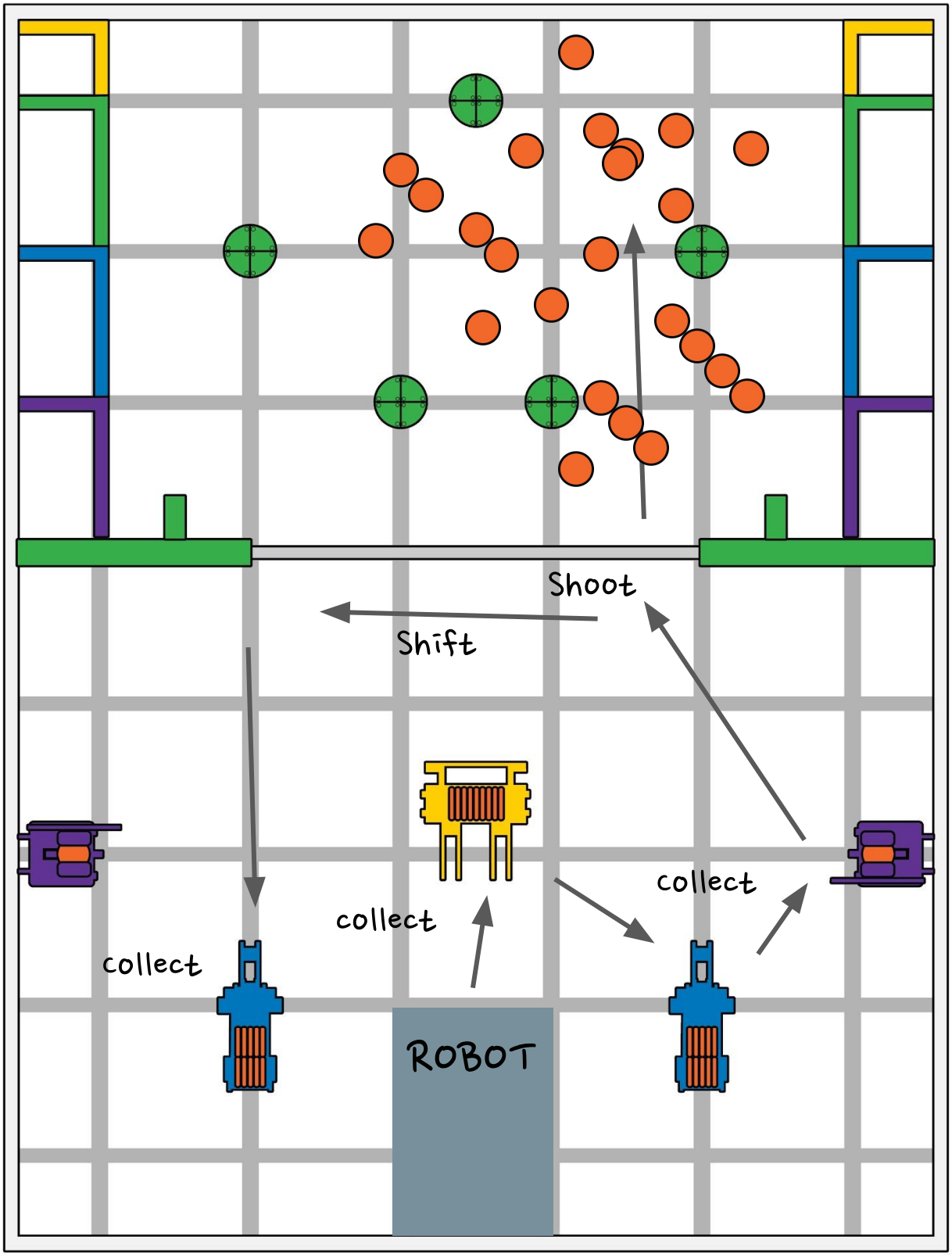
Competition Reflection:

Things we can work on:

- We were thinking of changing our robot's shooter to be able to collect pucks too, making it a dual-use mechanism.
- We also wanted to do some more driving practice to be able to earn more points faster.
- Next competition, we want to give teammates who didn't get to drive a chance to drive.
- We also want to have a team t-shirt ready for the next competition!

Project Successes at our first competition

Skills Strategy (As of Jan. 2023)

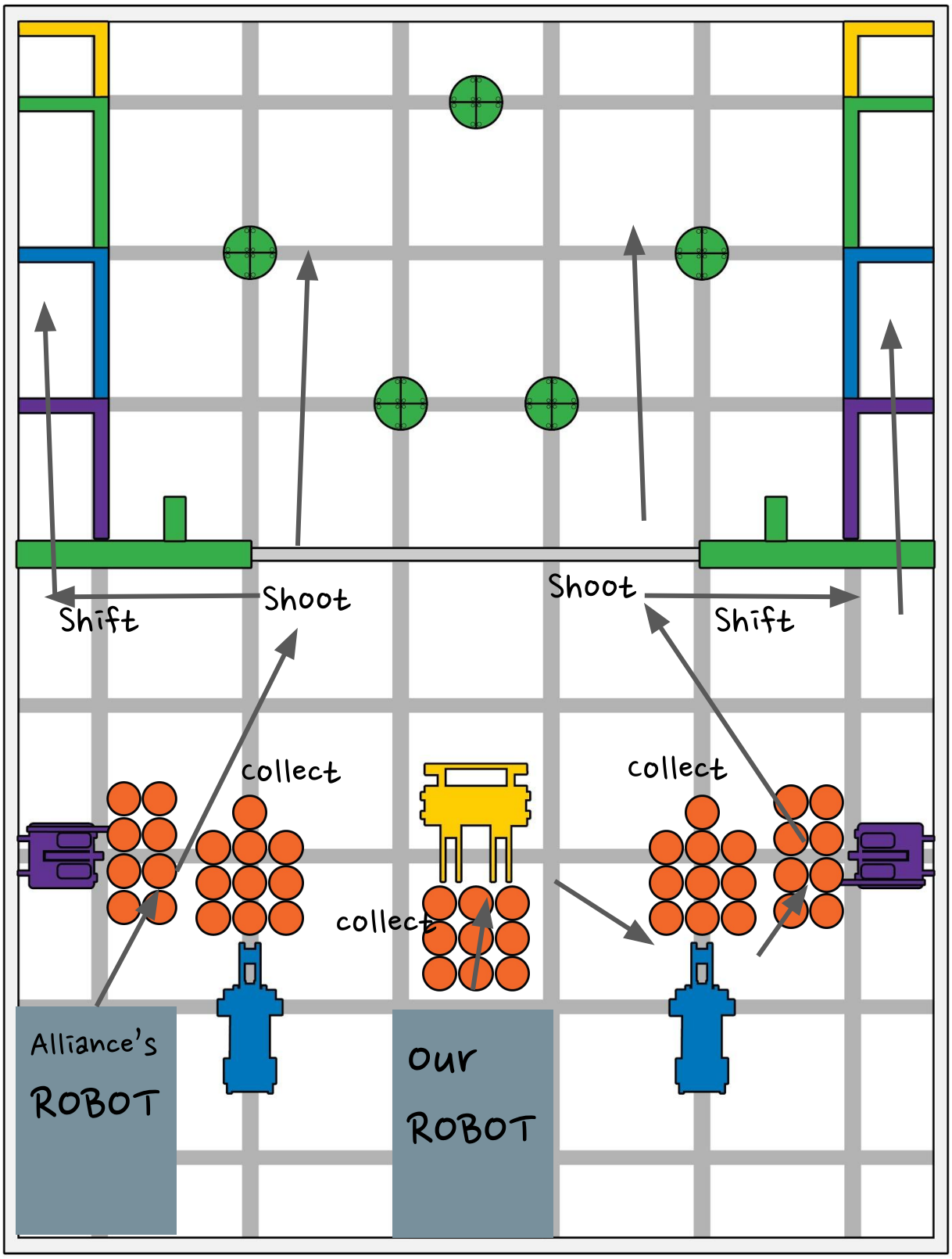


Project Skills Strategy 1.0

Name

Date January 2023

Teamwork Match Strategy (as of Jan. 2023)

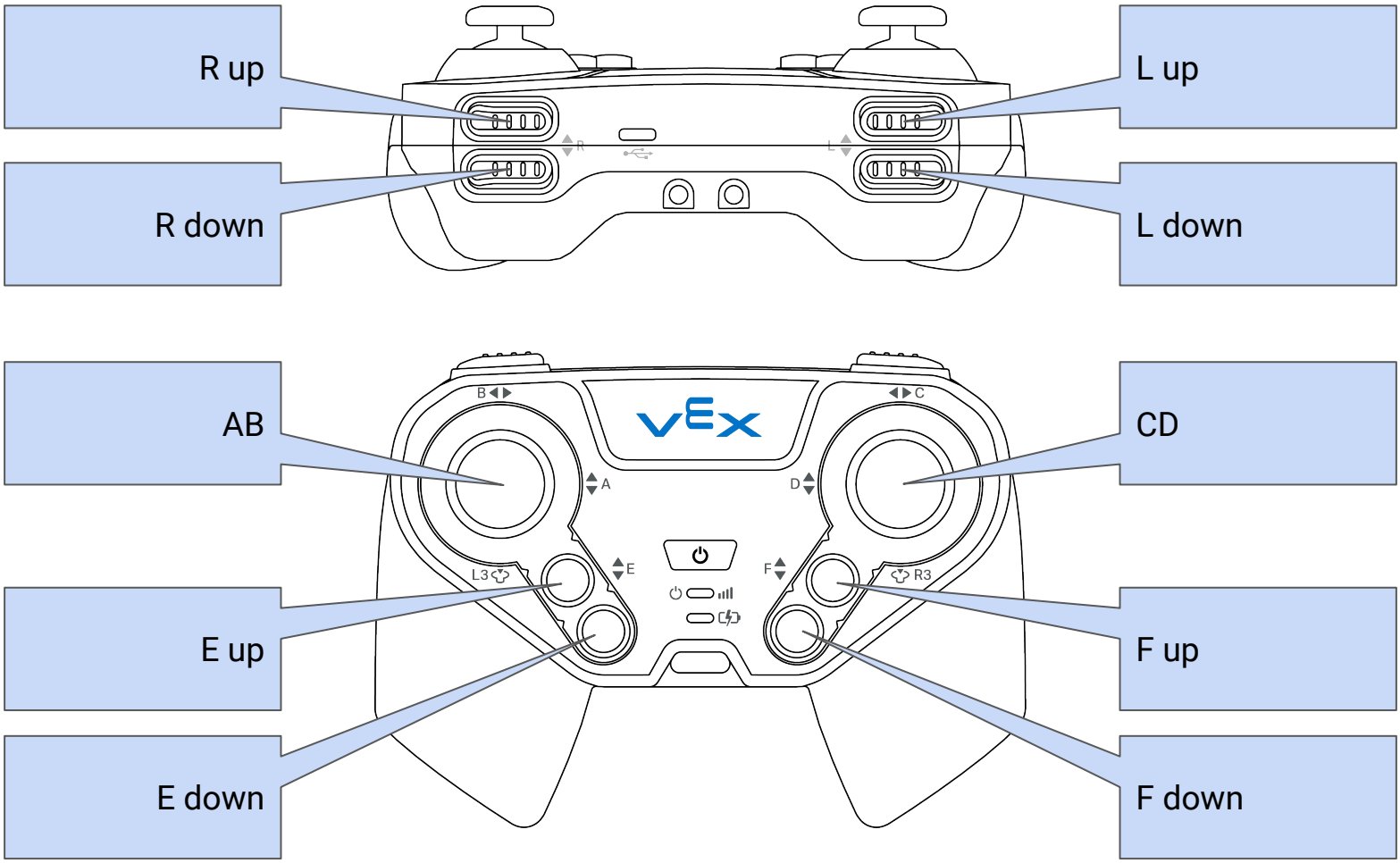


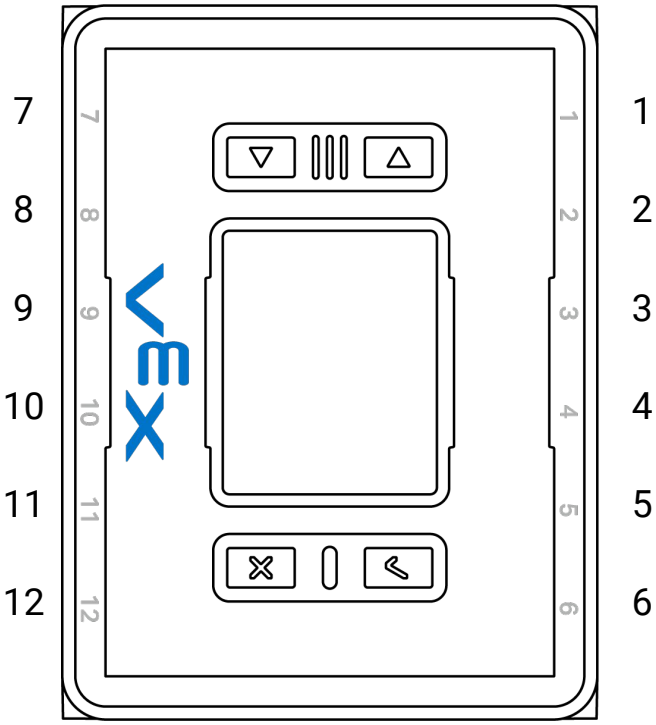
Project

Name

Date January 2023

The teamwork strategy was based on the alliance being able to collect the blue and purple towers, and having a touchdown mechanism. When we are actually at the competition (because every team's robot is different) we will change the strategy will also listening to our alliance's proposed strategy.





Project

Name Date Page 66

Gear Ratio

Introduction to Gear Ratio!

Why do we have Gear Ratio?

We have gear ratios to represent a smaller and easier version of our gear combinations. It can change the speed or strength of our pieces so that it is easily adjustable.

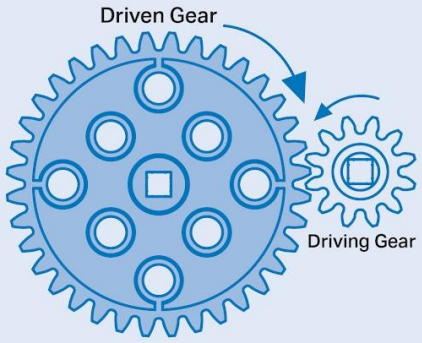
What makes it so Important?

Gear ratios are important as well as the overall gear idea. If we ever want to make the robot drive faster, we could change the gear ratio. Or if we ever wanted our arm to be stronger, we could change it's gear ratio.

Something we learned is that if you want to change the gear to go faster, it will also become slower. Vice versa, if you want to change the gear to bee faster, it will become weaker.

Gear Formulas

Gear Ratio = $\frac{\text{\# of Driven Gear Teeth (Output)}}{\text{\# of Driving Gear Teeth (Input)}}$



Power Transfer is a 1:1 gear ratio where the driving and driven gear have the **same number** of teeth.

Increasing Torque (lowering speed) is a gear ratio where the driving gear has **fewer teeth** than the driven gear.

Increasing Speed (lowering torque) is a gear ratio where the driving gear has **more teeth** than the driven gear.

Compound Gear Ratio = (Gear Ratio 1) × (Gear Ratio 2) × (...)

Motion Formulas

Average Speed = $\frac{\text{Total Distance}}{\text{Total Time}}$

Rotational Speed = $\frac{\text{\# of Turns}}{\text{Time}} = \frac{\text{Degrees}}{\text{Time}}$

Power = Force × Velocity

Torque = Force × Distance

Distance is from the **axis of rotation**

Circumference = π × Diameter

π ≈ 3.14

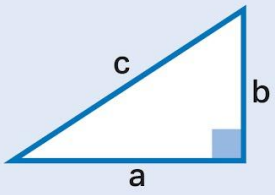
Force = Mass × Acceleration

Mathematical Formulas

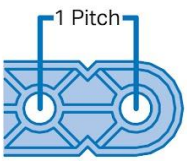
Complimentary angles are angles that sum to 90°

Supplementary angles are angles that sum to 180°

Pythagorean Theorem: $c^2 = a^2 + b^2$

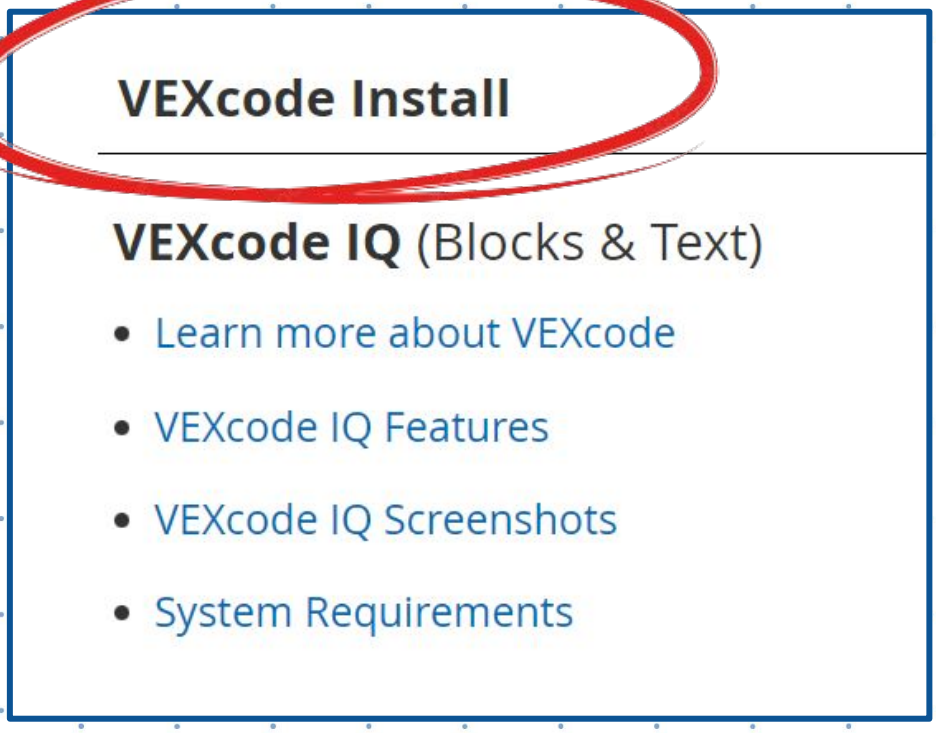


1 Pitch = 0.5 in =12.7 mm



Programming

Downloading the IDE

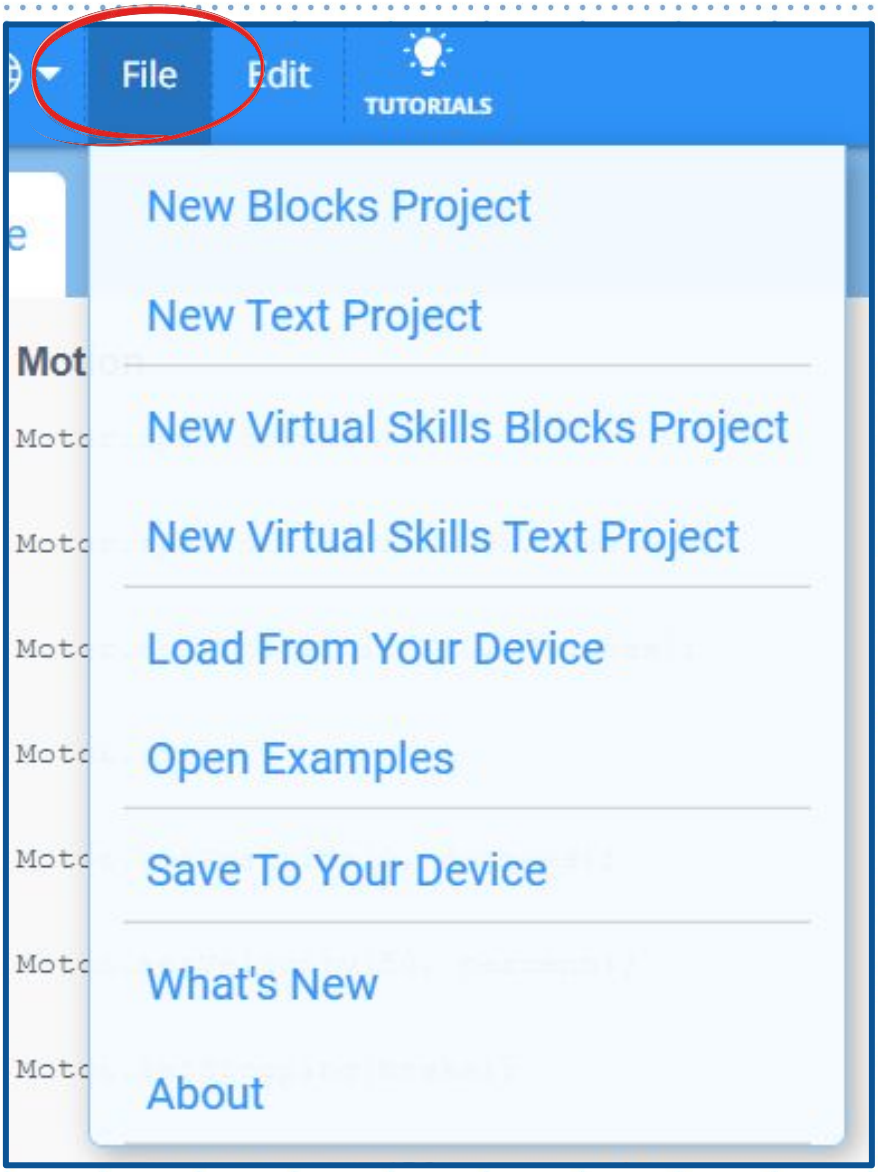


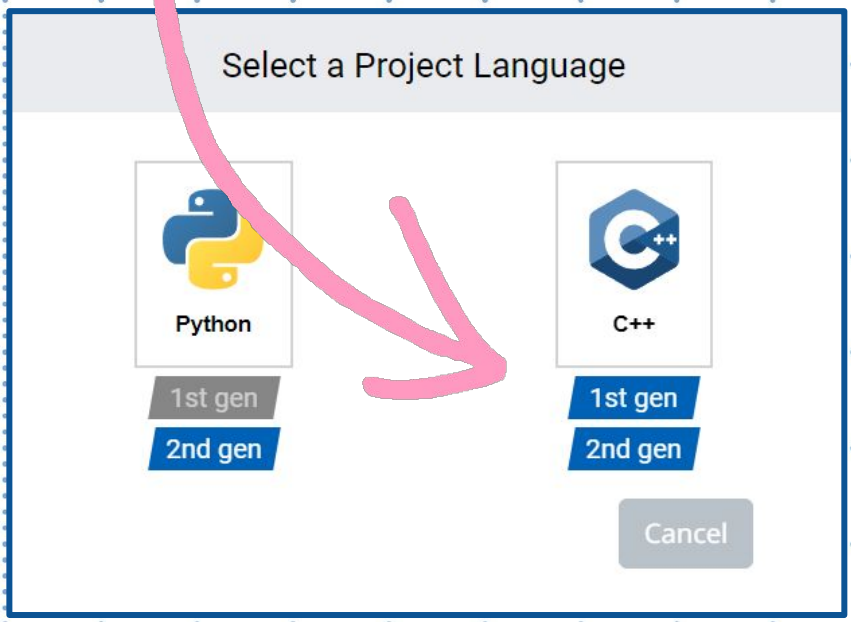
we learned that the new generation of pieces also came with a new IDE/ programming system. This is how we learned to download the new IDE.

The Steps:

1. First, we went to https://www.vexrobotics.com/vexcode/install/iq
2. we could either install the software, or use the online version. The online version is @ codeIQ.vex.com
3. After downloading, we could choose whether we want to use blocks or text (we decided on text, because that's what we've been using before).

4. To start a new project, we click "File", then "New Text Project"
5. We decided to program with C++ this year, because we've already learned a little bit of Python before. We wanted to try something new!



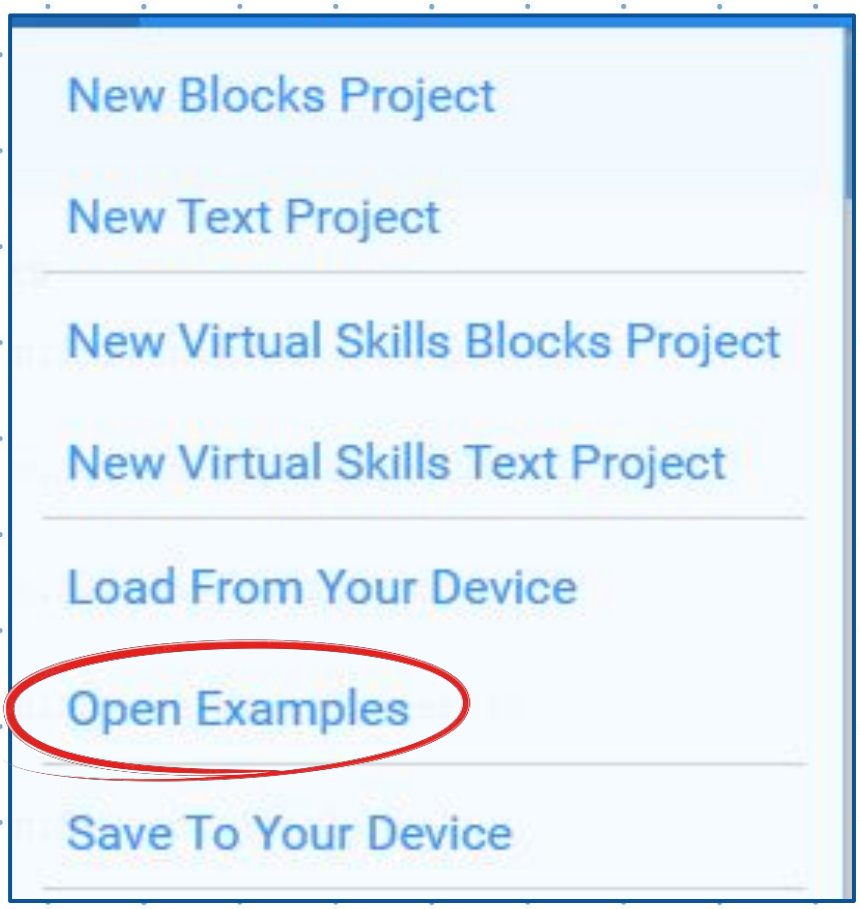


6. Congrats! You've just created a new file. Now, practice opening sample programs to see example programs!

Learn about Sample Programs on the next page!

Project More info on the IDE

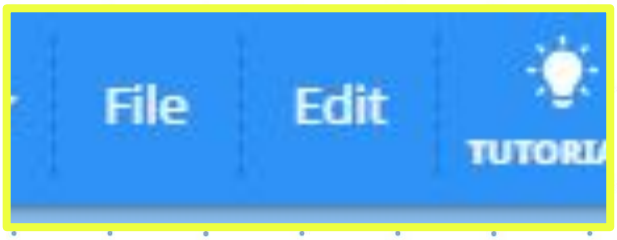
Using Sample Programs

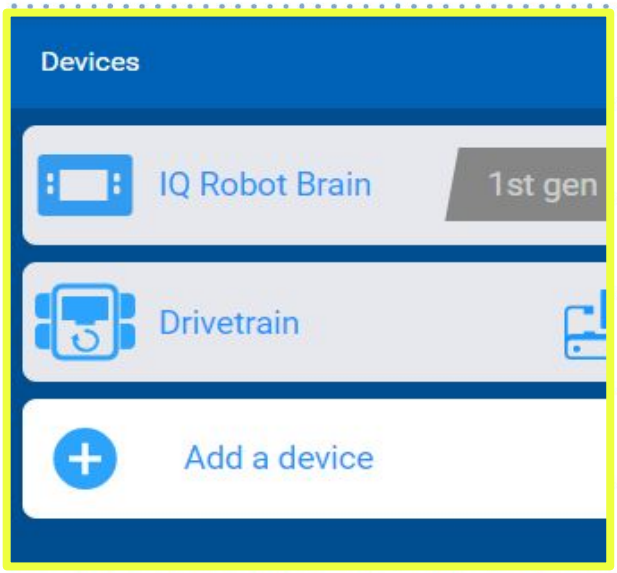


When we first opened the IDE, we learned that there were sample programs we could practice with. Sample programs will help us learn how to write programs in the new c++ language, and we can learn some basic commands. Looking at sample programs will give us a head start in programming.

Project Using Sample Programs for C++

What we did to run the program:





1. After opening https://codeiq.vex.com/ or open your download software, then click "File", then "Open Examples".
2. There are many examples we looked through, including basebot, clawbot, and snapshot. choose one.
3. To run the code, select a slot to place the program to by clicking the "slot" button at the top.
4. Name your motors. Press this icon:  at the top right corner. Then select "add a device" Add your motors and sensors.
5. Save the code by going to "File" and "Save to your device", then connect the brain and controller to the computer. Your Program should appear in the slot you selected.

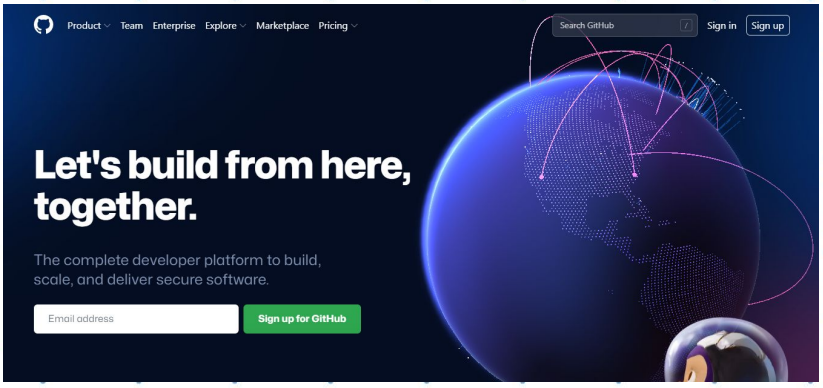
Figuring Out Github

This is the first year we are using GitHub. GitHub is a platform that allows us share our codes to teammates. It is attached to Git, which is a software that lets us save all copies of our code, and put to code into a repository.

Here is how we downloaded/prepared GitHub and Git:

1. First, we went to github.com, and created an account. Each of us created our own accounts.
2. Then, one of us created a repository, and invited the rest of us to be collaborators.

github.com



3. Next, we downloaded Git at https://git-scm.com/download
4. We created a new file in GitHub to practice linking Git and Github, and transferring files.

Project Github and Git

git

These are some basic commands we used in Git:

git init

git remote add origin URL_of_git_repository

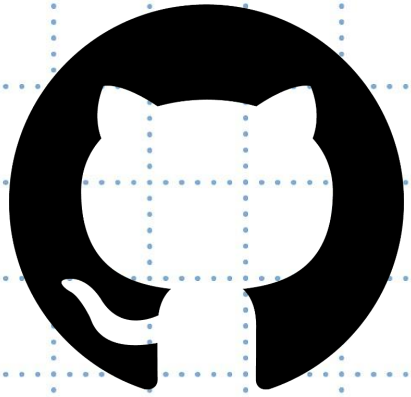
git pull origin master

git add .

git commit -m "message"

git remote -v

git push -u origin master



GitHub

Project Commands We Put into Git

What Do All the Commands Do?

git init
- Turns local directory into a git repository

git remote add origin URL_of_git_repository (make sure file explorer is open to the right folder)
- Connecting local repository to a remote repository on github

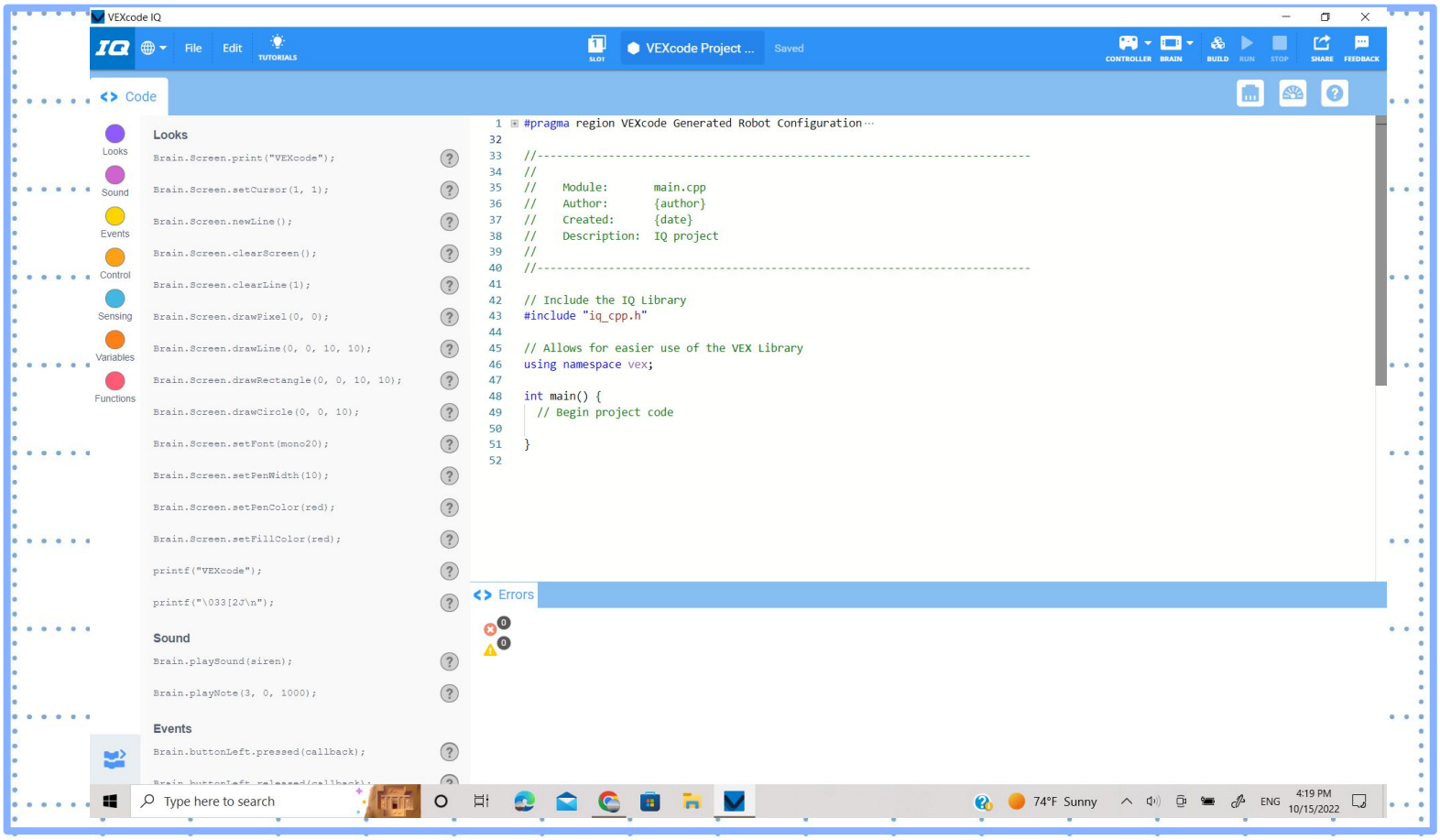
git pull origin master
- Allows to pull all files in remote github repository

git add .
- Checks all change into local git repository
- Puts into a staging area
- The "." means to add all files

git commit -m "message"
- Commits the changes

git remote -v
- Checks which github repository you are connected to

git push -u origin master
- Pushes change into github



On the meet of 10/15/22...

For this week's meet, we went through a sample code in the 'Gen 2' part of the updated code for c++. We went through the first couple lines showing the #Pragma sign, strings, and a while loop. While not knowing much knowledge, we tried to connect this code to the one last year in Robot c.

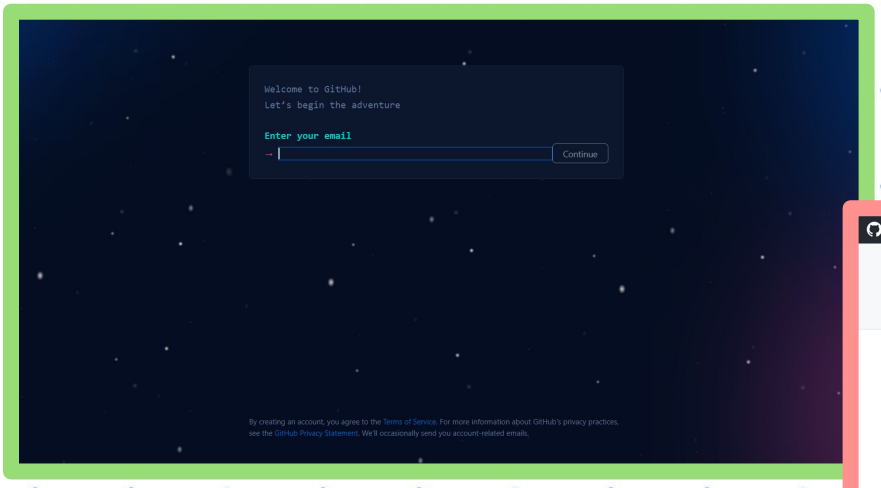
Project

Name Date Page 47

A Tour of Github!

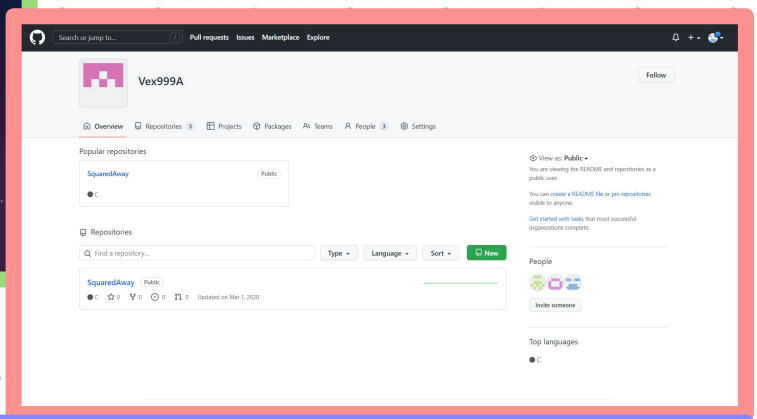
Step 1:

The first step is to make an account for github and make a fun username!



Step 2:

The second step is to finish making your account and get added by your team account or coach.



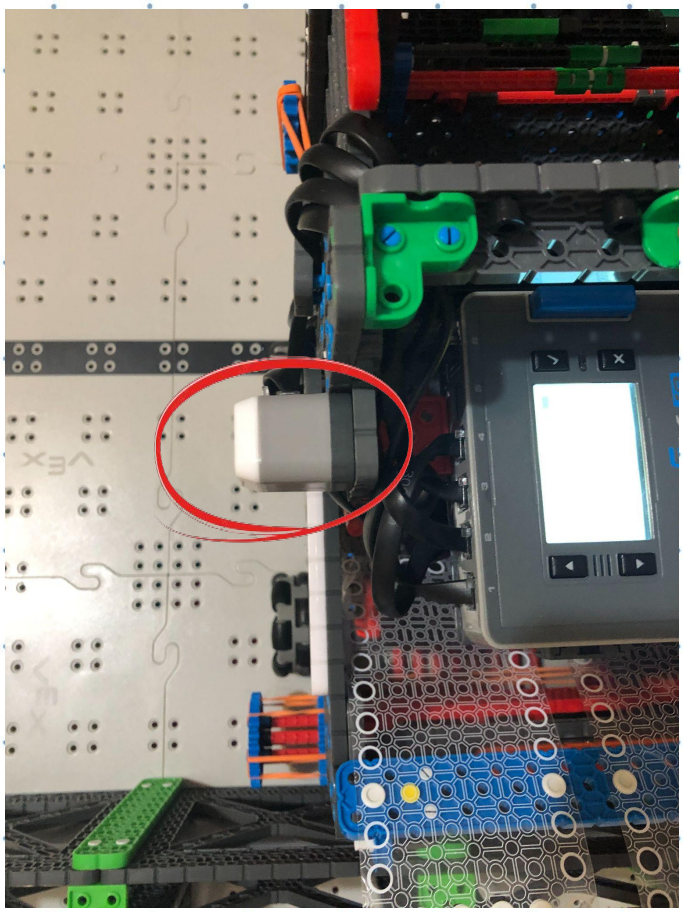
Step 3:

Post your code, and start sharing with your team!

Project Using Github!

Name Zoe Pak Date 10/31/22

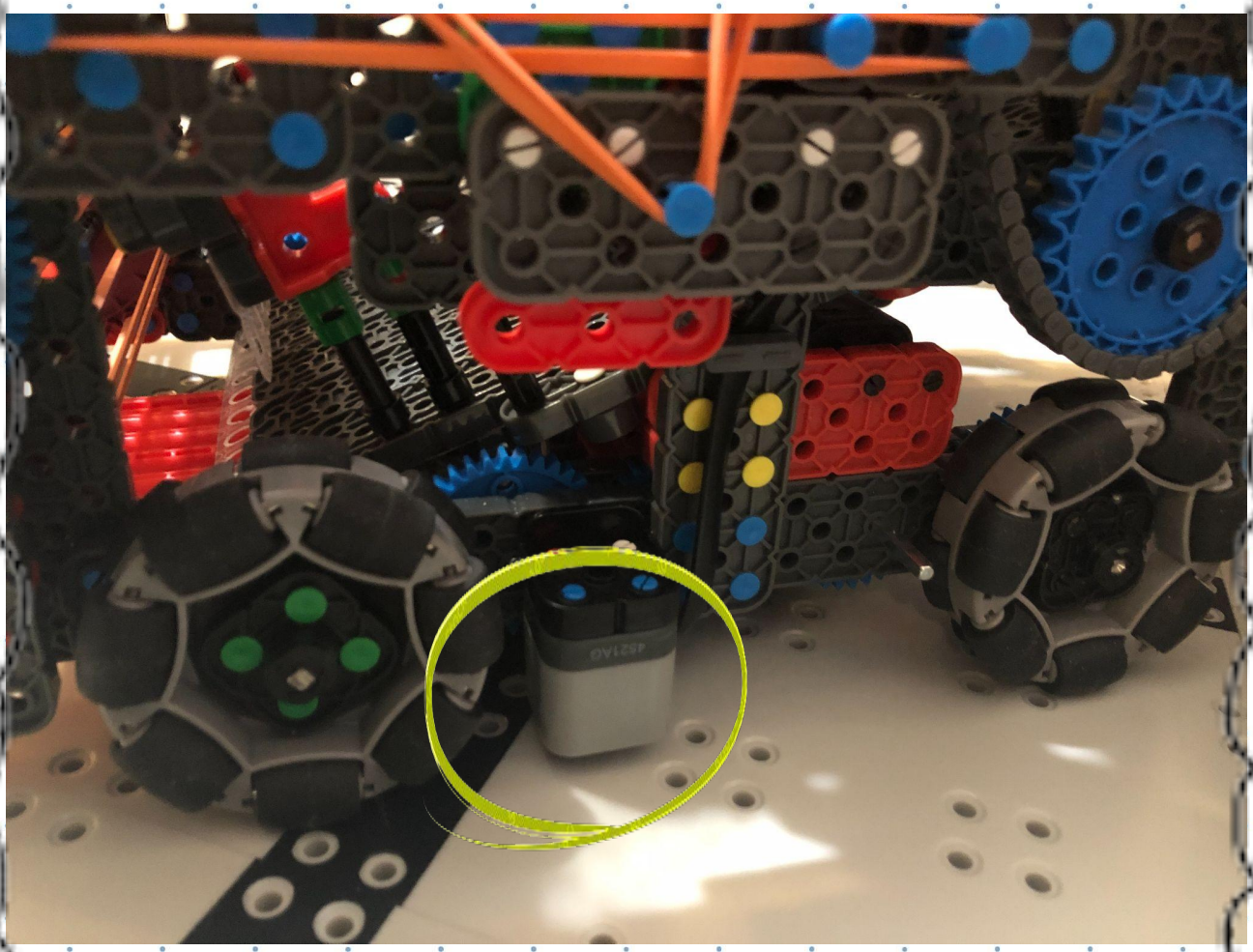
Using the LED sensor



The LED sensor is on the side of our robot. We use it when starting a program. When the red light goes on, it means the gyro sensor is still calibrating. When the light turns green, it means the program is ready to start. It's easier to start a program with the LED.

Project LED sensor

Using the COLOR sensor



The color sensor sees black and white. We use it to make sure the robot drives straight while driving. We programmed it to make the robot drive on the black line, and since the black line is straight, the robot will drive straight. This allows more accuracy.

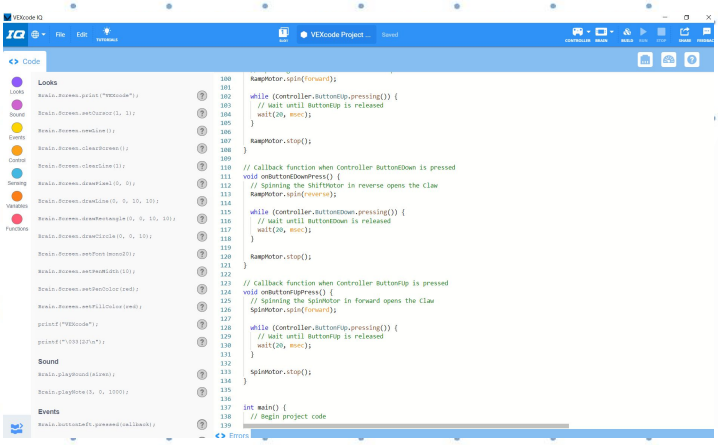
Project Color sensor

Why do we use TeleOps Programs?

What is TeleOps: TeleOps programs allow us to control the commands on the controller will driving in teamwork matches and skills. That way we can customize each button.

Why is this important: Being able to customize what each button does on a controller will make driving for drivers much easier. It will be more comfortable for the drivers because the controls will be changed based on our preferences.

How does it work: we made a program to program what button will do what and what happens when it is pressed. Then, we upload the program to the brain, run the program, and start driving.



Project What is TeleOps?

Name kristen Date 1/14 Page 82

Teleops in Programming

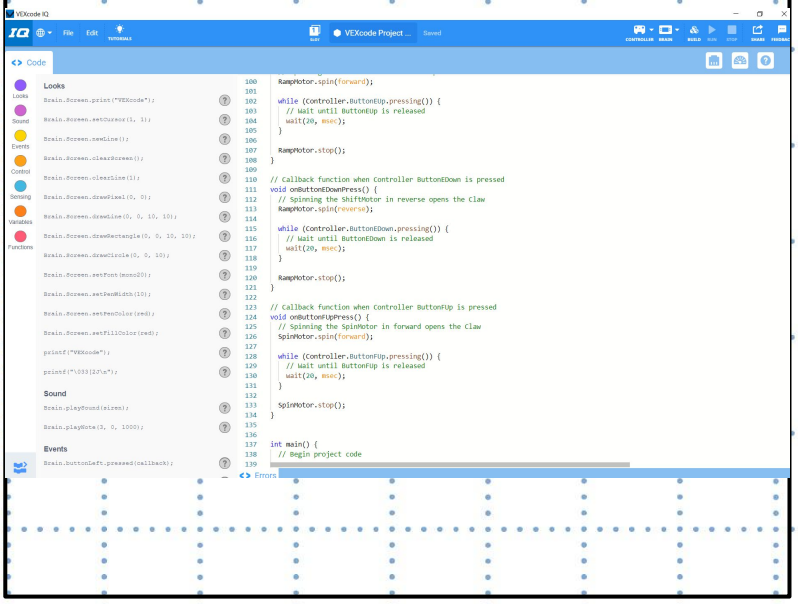
Goal:

- Be able to comfortably drive the robot no matter the driver.
- Switch controller buttons using programming

Problem:

- The program was not loading properly and the buttons seemed to not be working.

Picture:



Solution:

- We found out that we had two while loops in a row, but that could not work since the code is followed consecutively.
- We combined the while loops to run correctly in order.

Project Teleops in Programming

Name Zoe Pak Date 11/5/22 Page 49

TeleOps Program

```
#pragma config(Sensor, port2,  LED,          sensorVexIQ_LED)
#pragma config(Sensor, port8,  Gyro,         sensorVexIQ_Gyro)
#pragma config(Motor,  motor1,        Right,       tmotorVexIQ, PIDControl, driveRight, encoder)
#pragma config(Motor,  motor3,        Shift,       tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor4,        Ramp,        tmotorVexIQ, PIDControl, reversed, encoder)
#pragma config(Motor,  motor6,        Left,        tmotorVexIQ, PIDControl, reversed, driveLeft, encoder)
#pragma config(Motor,  motor9,        Tail,        tmotorVexIQ, PIDControl, reversed, encoder)
#pragma config(Motor,  motor10,       Shooter,     tmotorVexIQ, PIDControl, reversed, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

//https://www.vexforum.com/t/discussion-on-using-tasks-in-robotc/33025
//https://robocatz.com/functions-motor.htm
//https://github.com/vex999A/PitchingIn


/*Task to automatically rock the ramp*/
task AutoRockerTask()
{
	while(true)
	{
		/*Activate auto-rocker by channel F - Up & Down together*/
		if(getJoystickValue(BtnFUp) == 1)
		{
			if(getJoystickValue(BtnFDown) == 1){
				int counter = 0;
				while(counter < 5){
					//playSound(soundTada);
					//setMotorSpeed(Left, 100);
					//setMotorSpeed(Right, 100);
					//setMotorSpeed(Shooter, 100);
					//setMotorSpeed(Ramp, 100);
					//sleep(5000);
					//setMotorSpeed(Ramp, 0);
```

```
//Sleep(10);
                                    //SetMotorSpeed(Ramp, -100);
                                    //Sleep(5000);
                                    //SetMotorSpeed(Ramp, 0);
                                    //Sleep(10);
                                    counter++;
                                }
                            }
                        }
                    }
}

task main()
{
    /*************************
    *control channel Setup *
    *************************
    channel C/D: JoyStick Drive
    channel A: Forward/Backward only - reversed

    channel L: Shift operation
    channel R: Shooter operation

    channel E: Tail operation
    channel F: Ramp operation

    channel RUp + RDown: AutoRocker operation
    */

    /*******************
    *Declare variables*
    *******************/
    static const int threshold = 10; //Set threshold constant for Joystick input to weed out noise

    static const int forwardThrottle = 1; //Set forward throttle constant
    static const int turnThrottleN = 4; //Set turn throttle numerator constant
    static const int turnThrottleD = 5; //Set turn throttle denominator constant
```

```
static int forwardSpeed; //declare local static variable forwardSpeed
	static int turnSpeed; //declare local static variable turnSpeed

	/***************
	*Initialization*
	***************/
	setMotorEncoderUnits(encoderDegrees);

	setMotorBrakeMode(Right, motorBrake); //motorCoast, motorBrake, motorHold
	setMotorBrakeMode(Left, motorBrake);
	setMotorBrakeMode(Shift, motorBrake);
	setMotorBrakeMode(Shooter, motorCoast);
	setMotorBrakeMode(Ramp, motorHold);
	setMotorBrakeMode(Tail, motorBrake);

	resetMotorEncoder(Right);
	resetMotorEncoder(Left);
	resetMotorEncoder(Shift);
	resetMotorEncoder(Shooter);
	resetMotorEncoder(Ramp);
	resetMotorEncoder(Tail);

	/*Start auxiliary tasks*/
	startTask(AutoRockerTask);  // start task AutoRockerTask

	/*Start main control infinite loop*/
	while(true)
	{
		/*Drivetrain
		*Right joystick: we use the right joystick (both channel C and D) to control both driving
and turning,
		*Left joystick: we use the left joystick to do only straight movements (channel A for
straight forward and backward).
		*/
		//displayTextLine(3, "joystick: %d", getJoystickValue(chD));
		if(getJoystickValue(chC) > threshold  ||
			getJoystickValue(chC) < -threshold ||
		getJoystickValue(chD) > threshold  ||
		getJoystickValue(chD) < -threshold )
```

```
{
		forwardSpeed = getJoystickValue(chD)/forwardThrottle; //grab channel D Joystick
value and store in forwardSpeed
		turnSpeed = (getJoystickValue(chC)*turnThrottleN)/turnThrottleD; //grab channel C
Joystick value and store in turnSpeed
		setMotorSpeed(Left, forwardSpeed + turnSpeed);
		setMotorSpeed(Right, forwardSpeed - turnSpeed);
		//displayTextLine(3, "forward+turn: %d", forwardSpeed + turnSpeed);
		//displayTextLine(3, "joystick: %d", getJoystickValue(chD));
		sleep(10); //Pause to allow system to process other tasks
	}
	else if(
		getJoystickValue(chA) > threshold  ||
	getJoystickValue(chA) < -threshold )
	{
		forwardSpeed = getJoystickValue(chA)/forwardThrottle; //grab channel A Joystick
value and store in forwardSpeed
		setMotorSpeed(Left, -forwardSpeed);
		setMotorSpeed(Right, -forwardSpeed);
		sleep(10); //Pause to allow system to process other tasks
	}
	else
	{
		setMotorSpeed(Left, 0); //Set all drive motors to rest
		setMotorSpeed(Right, 0); //Set all drive motors to rest
	}


	/*Shift by channel E*/
	if(getJoystickValue(BtnEUp) == 1)
	{
		setMotorSpeed(Tail, 100);
		sleep(10); //Pause to allow system to process other tasks
	}
	else if(getJoystickValue(BtnEDown) == 1)
	{
		setMotorSpeed(Tail, -100);
		sleep(10); //Pause to allow system to process other tasks
	}
```

```
else
        {
                setMotorSpeed(Tail, 0);
        }


        /*Shift by channel L*/
        if(getJoystickValue(BtnLUp) == 1)
        {
                setMotorSpeed(Shift, 100);
                sleep(10); //Pause to allow system to process other tasks
        }
        else if(getJoystickValue(BtnLDown) == 1)
        {
                setMotorSpeed(Shift, -100);
                sleep(10); //Pause to allow system to process other tasks
        }
        else
        {
                setMotorSpeed(Shift, 0);
        }

        /*Ramp by channel F*/
        if(getJoystickValue(BtnFUp) == 1)
        {
                setMotorSpeed(Ramp, 100);
                sleep(10); //Pause to allow system to process other tasks
        }
        else if(getJoystickValue(BtnFDown) == 1)
        {
                setMotorSpeed(Ramp, -100);
                sleep(10); //Pause to allow system to process other tasks
        }
        else
        {
                setMotorSpeed(Ramp, 0);
        }
```

```
        /*Shooter
            *Button R_up for turning on shooter, and R_down for reversing and stopping the shooter.
            */
            if(getJoystickValue(BtnRUp) == 1)
            {
                setMotorSpeed(Shooter, 100);
                sleep(10); //Pause to allow system to process other tasks
            }
            else if(getJoystickValue(BtnRDown) == 1)
            {
                setMotorSpeed(Shooter, -100);
                sleep(10); //Pause to allow system to process other tasks
                setMotorSpeed(Shooter, 0);
            }

    }

}
```

Autonomous Program

```
#pragma config(Sensor, port2,  TouchLED,      sensorVexIQ_LED)
#pragma config(Sensor, port7,  color,         sensorVexIQ_colorGrayscale)
#pragma config(Sensor, port8,  Gyro,          sensorVexIQ_Gyro)
#pragma config(Motor,  motor1,        Right,        tmotorVexIQ, PIDControl, driveRight, encoder)
#pragma config(Motor,  motor3,        Shift,        tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor4,        Ramp,         tmotorVexIQ, PIDControl, reversed, encoder)
#pragma config(Motor,  motor6,        Left,         tmotorVexIQ, PIDControl, reversed, driveLeft, encoder)
#pragma config(Motor,  motor9,        Tail,         tmotorVexIQ, PIDControl, reversed, encoder)
#pragma config(Motor,  motor10,       Shooter,      tmotorVexIQ, PIDControl, reversed, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

//https://www.vexforum.com/t/discussion-on-using-tasks-in-robotc/33025
//https://robocatz.com/functions-motor.htm
//https://github.com/vex999A/PitchingIn


/*Task to automatically rock the ramp*/
task AutoRockerTask()
{
	while(true)
	{
		/*Activate auto-rocker by channel F - Up & Down together*/
		if(getJoystickValue(BtnFUp) == 1)
		{
			if(getJoystickValue(BtnFDown) == 1){
				int counter = 0;
				while(counter < 5){
					//playSound(soundTada);
					//setMotorSpeed(Left, 100);
					//setMotorSpeed(Right, 100);
					//setMotorSpeed(Shooter, 100);
					//setMotorSpeed(Ramp, 100);
					//sleep(5000);
					//setMotorSpeed(Ramp, 0);
					//sleep(10);
					//setMotorSpeed(Ramp, -100);
					//sleep(5000);
					//setMotorSpeed(Ramp, 0);
					//sleep(10);
					counter++;
				}
			}
		}
	}
}
```

```
task main()
{
	/*Initialization steps start*/
	setTouchLEDColor(TouchLED, colorRed); //Start initialization steps

	/*******************
	*Declare variables*
	*******************/
	int turning;
	int Rdown = -110;

	/****************
	*Initialization*
	****************/
	wait1Msec(5000);

	//Set motor encoder units
	setMotorEncoderUnits(encoderDegrees);

	//Set motor brake mode
	setMotorBrakeMode(Right, motorBrake); //motorCoast, motorBrake, motorHold
	setMotorBrakeMode(Left, motorBrake);
	setMotorBrakeMode(Shift, motorBrake);
	setMotorBrakeMode(Shooter, motorCoast);
	setMotorBrakeMode(Ramp, motorHold);
	setMotorBrakeMode(Tail, motorBrake);

	//reset sensors/encoders
	resetGyro(Gyro); //Reset gyro sensor to 0 reading

	resetMotorEncoder(Right);
	resetMotorEncoder(Left);
	resetMotorEncoder(Shift);
	resetMotorEncoder(Shooter);
	resetMotorEncoder(Ramp); //full up position; go down -110
	resetMotorEncoder(Tail); //full down position; go up 380
```

```
//calibrate gyro sensor
    startGyrocalibration(Gyro, gyrocalibrateSamples1024); // longer cal, works better
    wait1Msec(100);
    // wait for calibration to finish
    while(getGyrocalibrationFlag(Gyro))
    {
        displayTextLine(1, "calibrating... %02d");
        wait1Msec(100);
    }
    setTouchLEDcolor(TouchLED, colorGreen); //Gyro sensor calibration completed
    //playSound(soundTada);
    eraseDisplay();
    resetGyro(Gyro);
    /*initialization steps complete*/


    /*autonomous program starts here*/
    /*Start auxiliary tasks*/
    startTask(AutoRockerTask); //start AutoRockerTask task

    while(true)
    {
        /*Touch LED sensor to start the autonuous program*/
        waitUntil(getTouchLEDvalue(TouchLED) == 1);

        /**
        //testing testing
        setServoTarget(Tail, 380);
        wait1Msec(1000);
        setServoTarget(Tail, 0);
        wait1Msec(1000);
        setServoTarget(Tail, 380);
        wait1Msec(1000);

        setServoTarget(Ramp, Rdown);
        wait1Msec(1000);
        setServoTarget(Ramp, 0);
        wait1Msec(1000);
        setServoTarget(Ramp, Rdown);
        wait1Msec(1000);
        setServoTarget(Ramp, 0);
        **/
```

```
//lift up Tail
setServoTarget(Tail, 380);

//back up
resetMotorEncoder(Left);
resetMotorEncoder(Right);
setMotorTarget(Left, -1000, 50);
setMotorTarget(Right, -1000, 50);
waitUntilMotorStop(Left);
waitUntilMotorStop(Right);
wait1Msec(500);

//tilt up ramp to collect pucks from yellow repo
setServoTarget(Ramp, -40);
wait1Msec(500);
//drive forward
resetMotorEncoder(Left);
resetMotorEncoder(Right);
setMotorTarget(Left, 60, 50);
setMotorTarget(Right, 60, 50);
waitUntilMotorStop(Left);
waitUntilMotorStop(Right);
wait1Msec(500);
//down ramp
setServoTarget(Ramp, 0);
wait1Msec(500);
//back up a little
resetMotorEncoder(Left);
resetMotorEncoder(Right);
setMotorTarget(Left, -80, 100);
setMotorTarget(Right, -80, 100);
waitUntilMotorStop(Left);
waitUntilMotorStop(Right);
wait1Msec(50);
//come out of yellow repo
resetMotorEncoder(Left);
resetMotorEncoder(Right);
setMotorTarget(Left, 220, 100);
setMotorTarget(Right, 220, 100);
waitUntilMotorStop(Left);
waitUntilMotorStop(Right);
wait1Msec(50);
```

```
//Shift left
		resetMotorEncoder(Shift);
		setMotorTarget(Shift, -500, 100);
		waitUntilMotorStop(Shift);
		wait1Msec(500);

		//back up
		resetMotorEncoder(Left);
		resetMotorEncoder(Right);
		setMotorTarget(Left, -420, 80);
		setMotorTarget(Right, -420, 80);
		waitUntilMotorStop(Left);
		waitUntilMotorStop(Right);
		wait1Msec(500);

		//Shift left
		resetMotorEncoder(Shift);
		setMotorTarget(Shift, -300, 100);
		waitUntilMotorStop(Shift);
		wait1Msec(500);

		//turn 180 degree left
		resetGyro(Gyro);
		while(getGyroDegrees(Gyro) < 178)
		{
			turning = getGyroDegrees(Gyro);
			setMotorSpeed(Left, -(100-turning/2));
			setMotorSpeed(Right, (100-turning/2));
		}
		setMotorSpeed(Left, 0);
		setMotorSpeed(Right, 0);
		wait1Msec(50);

		//back up towards blue repo
		resetMotorEncoder(Left);
		resetMotorEncoder(Right);
		setMotorTarget(Left, -235, 100);
		setMotorTarget(Right, -235, 100);
		waitUntilMotorStop(Left);
		waitUntilMotorStop(Right);
		wait1Msec(500);
```

```
//drive forward slightly
resetMotorEncoder(Left);
resetMotorEncoder(Right);
setMotorTarget(Left, 40, 50);
setMotorTarget(Right, 40, 50);
waitUntilMotorStop(Left);
waitUntilMotorStop(Right);
wait1Msec(500);

//bring down Tail to collect pucks from blue repo
setServoTarget(Tail, 0);
wait1Msec(2000);

//drive forward slightly
resetMotorEncoder(Left);
resetMotorEncoder(Right);
setMotorTarget(Left, 200, 80);
setMotorTarget(Right, 200, 80);
waitUntilMotorStop(Left);
waitUntilMotorStop(Right);
wait1Msec(500);

//lift up Tail
setServoTarget(Tail, 380);

//Shift left
resetMotorEncoder(Shift);
setMotorTarget(Shift, -250, 100);
waitUntilMotorStop(Shift);
wait1Msec(500);

//drive forward
resetMotorEncoder(Left);
resetMotorEncoder(Right);
setMotorTarget(Left, 700, 100);
setMotorTarget(Right, 700, 100);
waitUntilMotorStop(Left);
waitUntilMotorStop(Right);
wait1Msec(500);
```

```
//turn on shooter
setMotorSpeed(Shooter, 50);

//rock the ramp
setServoTarget(Ramp, Rdown);
wait1Msec(800);
setServoTarget(Ramp, 0);
wait1Msec(800);
setServoTarget(Ramp, Rdown);
wait1Msec(800);
setServoTarget(Ramp, 0);
wait1Msec(800);
setServoTarget(Ramp, Rdown);
wait1Msec(800);
setServoTarget(Ramp, 0);
wait1Msec(800);
setServoTarget(Ramp, Rdown);
wait1Msec(800);
//bring down Tail to release pucks from upper reservoir
setServoTarget(Tail, 0);
wait1Msec(500);
setServoTarget(Ramp, 0);
wait1Msec(800);
setServoTarget(Ramp, Rdown);
wait1Msec(800);
setServoTarget(Ramp, 0);
wait1Msec(800);
setServoTarget(Ramp, Rdown);
wait1Msec(800);
setServoTarget(Ramp, 0);
wait1Msec(800);
setServoTarget(Ramp, Rdown);
wait1Msec(800);
setServoTarget(Ramp, 0);
wait1Msec(800);
setServoTarget(Ramp, Rdown);
wait1Msec(800);

//turn off shooter
setMotorSpeed(Shooter, 0);
//lift up Tail
setServoTarget(Tail, 380);
```

```
/**
		//back up from fence
		resetMotorEncoder(Left);
		resetMotorEncoder(Right);
		setMotorTarget(Left, -100, 100);
		setMotorTarget(Right, -100, 100);
		waitUntilMotorStop(Left);
		waitUntilMotorStop(Right);
		wait1Msec(50);
		**/

		//Shift left
		resetMotorEncoder(Shift);
		setMotorTarget(Shift, -800, 100);
		waitUntilMotorStop(Shift);
		wait1Msec(500);

		//back up from fence
		resetMotorEncoder(Left);
		resetMotorEncoder(Right);
		setMotorTarget(Left, -200, 100);
		setMotorTarget(Right, -200, 100);
		waitUntilMotorStop(Left);
		waitUntilMotorStop(Right);
		wait1Msec(50);

		//use color sensor to detect black line
		int threshold = 80; //white ~ 160; black ~ 50 (gray scale)
		while(getcolorvalue(color) > threshold) // While the color sensor in 'grayscale' mode reads a value
less than threshold
		{
			setMotorSpeed(Shift, -100);
		}
		setMotorSpeed(Shift, 0);
		wait1Msec(50);
```

```
//lower ramp
        setServoTarget(Ramp, 0);

        //Shift left
        resetMotorEncoder(Shift);
        setMotorTarget(Shift, -120, 100);
        waitUntilMotorStop(Shift);
        wait1Msec(500);

        //back up to blue repo
        resetMotorEncoder(Left);
        resetMotorEncoder(Right);
        setMotorTarget(Left, -700, 100);
        setMotorTarget(Right, -700, 100);
        waitUntilMotorStop(Left);
        waitUntilMotorStop(Right);
        wait1Msec(50);

        //drive forward slightly
        resetMotorEncoder(Left);
        resetMotorEncoder(Right);
        setMotorTarget(Left, 40, 50);
        setMotorTarget(Right, 40, 50);
        waitUntilMotorStop(Left);
        waitUntilMotorStop(Right);
        wait1Msec(500);

        //bring down Tail to collect pucks from blue repo
        setServoTarget(Tail, 0);
        wait1Msec(2000);

        //drive forward slightly
        resetMotorEncoder(Left);
        resetMotorEncoder(Right);
        setMotorTarget(Left, 200, 80);
        setMotorTarget(Right, 200, 80);
        waitUntilMotorStop(Left);
        waitUntilMotorStop(Right);
        wait1Msec(500);
```

```
//lift up Tail
setServoTarget(Tail, 380);

//Shift right
resetMotorEncoder(Shift);
setMotorTarget(Shift, 300, 100);
waitUntilMotorStop(Shift);
wait1Msec(500);

//drive forward
resetMotorEncoder(Left);
resetMotorEncoder(Right);
setMotorTarget(Left, 620, 100);
setMotorTarget(Right, 620, 100);
waitUntilMotorStop(Left);
waitUntilMotorStop(Right);
wait1Msec(500);

//turn on shooter
setMotorSpeed(Shooter, 60);

//rock the ramp

setServoTarget(Tail, 0);
setServoTarget(Ramp, Rdown);
wait1Msec(800);
setServoTarget(Ramp, 0);
wait1Msec(800);
setServoTarget(Ramp, Rdown);
wait1Msec(800);
setServoTarget(Ramp, 0);
wait1Msec(800);
setServoTarget(Ramp, Rdown);
wait1Msec(800);
setServoTarget(Ramp, 0);
wait1Msec(800);
setServoTarget(Ramp, Rdown);
wait1Msec(800);
```

```
setMotorSpeed(Shooter, 0);

//back up
resetMotorEncoder(Left);
resetMotorEncoder(Right);
setMotorTarget(Left, -200, 100);
setMotorTarget(Right, -200, 100);
waitUntilMotorStop(Left);
waitUntilMotorStop(Right);
wait1Msec(500);

//Shift left
resetMotorEncoder(Shift);
setMotorTarget(Shift, -850, 100);
waitUntilMotorStop(Shift);
wait1Msec(1000);

//drive forward
resetMotorEncoder(Left);
resetMotorEncoder(Right);
setMotorTarget(Left, 300, 100);
setMotorTarget(Right, 300, 100);
waitUntilMotorStop(Left);
waitUntilMotorStop(Right);
wait1Msec(500);

/**
//Shift right
resetMotorEncoder(Shift);
setMotorTarget(Shift, 800, 100);
waitUntilMotorStop(Shift);
wait1Msec(1000);

//turn slight left
resetGyro(Gyro);
while(getGyroDegrees(Gyro) > -8)
```

```
{
            turning = getGyroDegrees(Gyro);
            setMotorSpeed(Left, (80+turning/2));
            setMotorSpeed(Right, -(80+turning/2));
        }
        setMotorSpeed(Left, 0);
        setMotorSpeed(Right, 0);
        wait1Msec(50);

        //drive to fence
        resetMotorEncoder(Left);
        resetMotorEncoder(Right);
        setMotorTarget(Left, 100, 100);
        setMotorTarget(Right, 100, 100);
        waitUntilMotorStop(Left);
        waitUntilMotorStop(Right);
        wait1Msec(50);
        **/
    }
}
```

Parts List

Parts List:

5x Wheels

6x motors

1 brain

12 1x13 beams

Standoffs

- 1x1
- 1x2
- 30x4
- 1x8

1 battery

1 touch LED sensor

Transparent plastic plates

5 1x11 beams

Rubberbands

6 2x20 beams

Gears

Axles

Peg connector

Wire

8 2x18

2 2x20

Spacers

Pegs without cap

Pegs with cap

4 1x8 beams

10 2x12 beams

6 1x11 beams

4 4x8 planks

2 2x2 planks

1x2 connector pieces

1x1 connector pieces

Horsey connector pieces

14 Stoppers

Washers

2x2 connector pieces

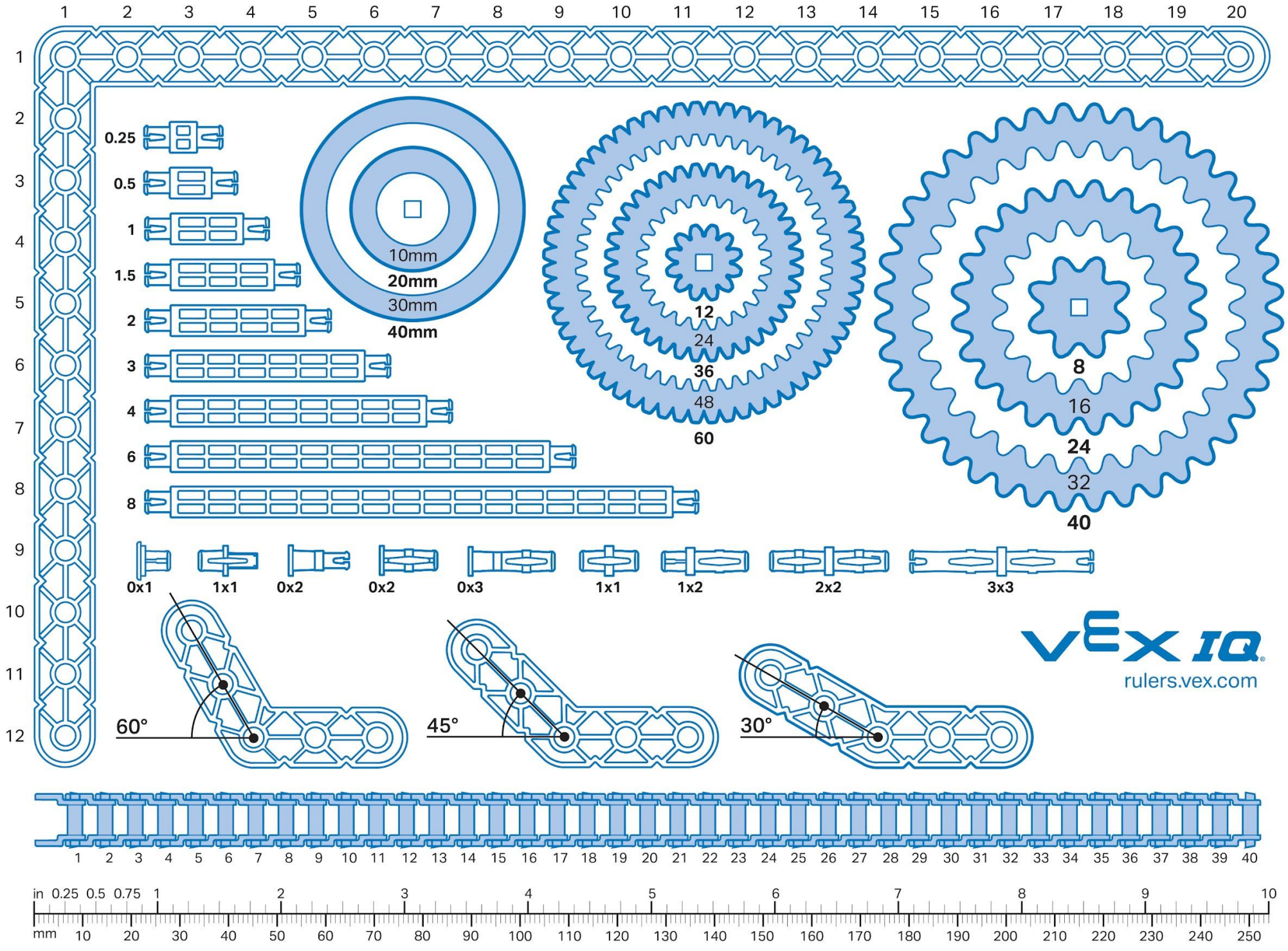
2x2 pegs

"L" pieces

"T" pieces

2x2 beams

1x4 beams


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12
0.25
0.5
1
1.5
2
3
4
6
8
10mm
20mm
30mm
40mm
12
24
36
48
60
8
16
24
32
40
0x1
1x1
0x2
0x2
0x3
1x1
1x2
2x2
3x3
60°
45°
30°
VEX IQ
rulers.vex.com
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
in 0.25 0.5 0.75 1 2 3 4 5 6 7 8 9 10
mm 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250