ENGINEERING JOURNAL

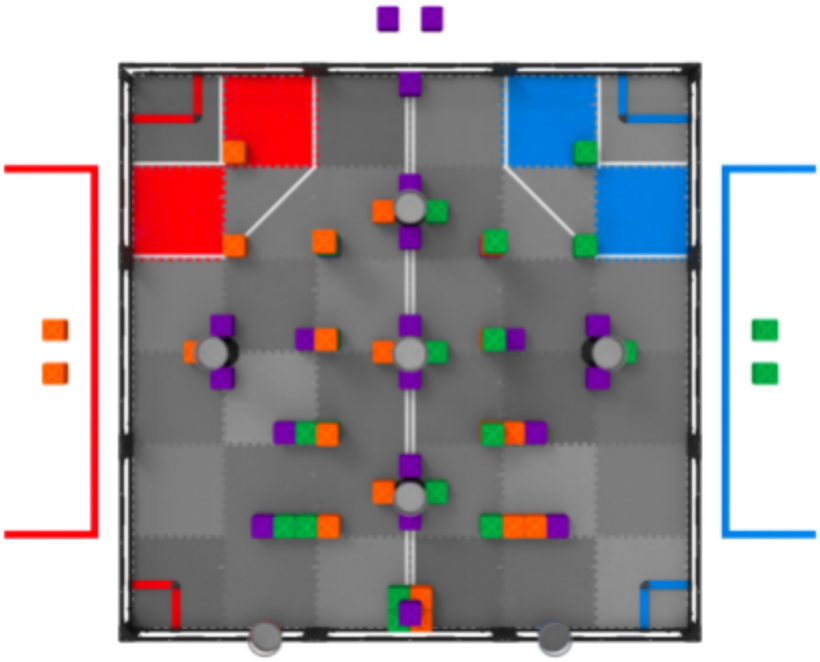
3818 A – Mad Max

International School Manila
Manila, Philippines

Jason Cruz
Justin Shin
John Abanes
Eion Chua
Alexi Nieminen

TABLE OF CONTENTS

INTRODUCTION



Game Analysis

This year's Vex Robotics Competition, Tower Takeover, requires two pairs of robots to complete certain tasks in order to score as many points as possible for their respective alliances. Scoring in this game simply requires stacking blocks in goals. However, what makes this game unique is the dynamic point multiplier system. Placing blocks in towers increases the value of all the other blocks of that color within the alliance's goal. Although, it must be noted that placing blocks in the towers not only increases your blocks' point values, but your opponents' blocks' point values as well. As such, not only do the alliances need to seek the optimal balance between scoring in towers and obtaining multipliers, but they also need to maximize their benefits from the aforementioned multipliers while minimizing their opponents' gains from them as well.

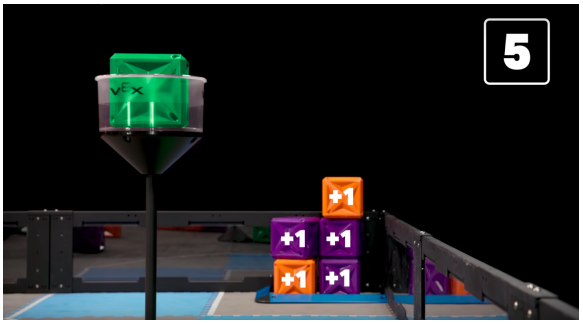
The match starts with a 15-second autonomous period, in which the robot cannot be controlled by a driver, but instead by a pre-written code. Afterwards, the game progresses to the driver-control period, in which the two alliances aim to score as much as possible. The winner of the autonomous round earns their alliance two blocks which they can place into the field at any time during the driver-control round, as well as 6 extra points.

Apart from matches with other alliances, there are also Skills Matches, in which the robot must score as much as possible within the field without competing robots in the arena. In these matches, the robot can score in any of the goals and place blocks on any of the towers. Skills matches have 1 minute driver and 1 minute autonomous rounds.

Game Pieces

- 4 Capture Zones
    - 2 per team
- 66 Cubes
    - 22 orange cubes
    - 22 green cubes
    - 22 purple cubes
- 7 Towers
    - 5 Neutral Towers
    - 2 Alliance-Specific Towers

Point System



Cube Color	Number of Cubes	Point Multiplier	Points after multiplier
Green	0	x2	0
Purple	3	x1	3
Orange	2	x1	2
Total Points			**5**

Cube Color	Number of Cubes	Point Multiplier	Points after multiplier
Green	0	x1	0
Purple	3	x2	6
Orange	2	x1	2
Total Points			**8**

Important Match Rules

- There is no limit to how far a Robot can expand once the match begins
- Robots must start touching the wall on their alliance side
- Robots cannot cross the middle autonomous line during the autonomous period, or the autonomous bonus is forfeited to the other alliance
- Robots must be preloaded with a cube before the match, touching the robot
- There is no limit to the number of cubes Robots can possess at any time
- Alliance towers can only be used by the alliance of that color

DESIGN OPPORTUNITY

The Team

				
Jason Cruz	Justin Shin	Alexi Nieminen	John Abanes	Eion Chua
Philippines	South Korea	Finland	Philippines	Philippines
- Builder - Primary Driver	- Builder - Quality Control - Researcher	- Secondary Programmer - Documenter - Data Analysis - Scouter	- Primary Programmer - Strategist	- Builder - Match Coach - Strategist - Secondary Driver

Jason (Senior): Having been a Driver since freshman year and a member of the Philippine Robotics Team, Jason boasts the most experience with piloting the robot for the seasons of Starstruck, In the Zone, and Turning point. His driving skills honed from playing racing video games. With an interest in pursuing architecture in college, his interest in building and construction allows him to transfer his passions. He is the primary illustrator for schematics and models.

Justin (Senior): With a strong interest in the sciences, Justin shows keen attention to detail when building. His passion for learning allows him to be an excellent researcher, and his specialties in Physics have allowed for the implementation of key ideas and use of problem solving skills to address issues and challenges that arise.

Alexi/Seba (Senior): Having been in the team since 9th grade, Seba holds experience in the way of competing in Vex Robotics. A natural resource manager, he works as the main documenter and analyser of the team, providing us with materials outside of those available in the classroom and helps us manage schedules to stay on task.

Eion (Junior): A certified Math genius, Eion has competed in dozens of international math competitions and olympiads, and has finished the IB HL Mathematics course 2 years in advance. Due to the nature of score multipliers in this year's game, his quick arithmetic skills would allow him to be a great coach and strategist both on and off the field, providing real-time decisions as point values change during the match.

John (Junior): John's proficiency in coding has allowed for a swift transition into VexCode from RobotC. His prior experience with C++ translated perfectly with the new software, and has allowed us to achieve complex tasks and functions that wouldn't be possible otherwise. Further, his code-based problem solving skills would give the potential for the robot to perform as best as possible.

Our diverse and unique backgrounds allow us to input ideas and see things at different viewpoints, helping us solve problems and make improvements along the way.

Design Problem

There are two main tasks to be completed in order to score points: Stacking cubes and scoring cubes on towers. It is extremely important to consider all possible aspects of the competition that a robot would be able to do. For a robot to be successful, it is necessary to strategise certain tasks to focus on, gearing the robot towards the most efficient way to score points. All cubes start with a value of 1 point, but get an additional point in value when the color of that cube is in a tower. As such, different structures are needed for the robot to accomplish certain tasks of the competition. When focusing on scoring points, a cube stacking mechanism is required to score cubes in each goal zone, while a vertical extension is required for a robot to score on towers of different sizes. Unlike previous years, there is no bonus at the end of a match. Therefore it is important to focus on maximizing scoring capability.

Constraints during construction include:

- Robot must fit within 18” x 18” x 18”
- Robot must be made of Vex parts only
    - We are limited to the parts we have available at our school’s robotics lab, resources that are split between other teams as well
    - We are working with the new V5 system, and are thus limited to V5 hardware components
- Robot must have 8 V5 motors maximum (6 with pneumatics)
- Time constraint of Sept. 2019 for completed base, and Oct. 2019 for completed robot + autonomous (for in depth dates, refer to Key Dates of Competition)

Design Brief

As a team, we are going to be designing, building, coding, and driving a robot that can efficiently balance and complete the most tasks of the competition (scoring and multiplying cubes) in order to efficiently score as much points as possible.

Strategies

There are only three different ways to score points during the match. These include:

- Stacking cubes in a goal zone
- Placing cubes in a tower, thus adding value to scored cubes
- Winning the autonomous bonus (which involves the two above)

What makes this game interesting is that the block tower multipliers not only increase the value of the blocks in your alliance’s goals, but the blocks in your opponents’ goals as well. Put simply, there are 3 main viable strategies. The first one is to focus on scoring stacks of blocks as high as possible in the goals while not putting much focus on the tower multipliers. The second one is to score tall stacks of blocks in

goals while also placing blocks in towers. The third one is to prevent the opponent from scoring while letting the other member of your alliance score points.

As a team, we've agreed that using the first strategy, which focuses on the ability to stack cubes in a goal zone, was the most viable in the early rounds, as this ability can be applied to both matches and skills. While reaching for high center towers was also important, these cubes would not be valid unless cubes were actually scored completely in the goals. Therefore, focusing on consistent block stacking already makes a win over the other alliance likely, since during the early rounds, many of the teams can't form high stacks nor place blocks on towers yet. However, as we progress to the more selective and competitive rounds, it is likely that we would have to switch to strategy 2. When many of the teams can now form high stacks comparable to ours, it now becomes important to maximize the value of each of our blocks in the goal stacks. It is important to note that the profit gained from placing a block of a certain color on towers does not directly depend on the number of blocks of that color within your goals, but rather how many *more* blocks of that color you have compared to the other team. The opponents can reduce this gap by using a roughly equal number of blocks from each color compared to your alliance. However, this is difficult to do in the fast-paced action of the game. As such, it is important to rapidly gauge the opponents' block stack composition with relation to your block stack composition, and with that information, decide on what block colors are profitable to be placed on towers. The last strategy, which is to block off the goals of the other alliance may be a viable strategy and would be very frustrating to go against. However, these robots usually have to sacrifice most of their scoring and movement capabilities for this. As such, it is very risky, since if the wall fails to block a scoring zone, much of the scoring potential of that alliance would be lost.

BUILDING THE ROBOT

Brainstorm

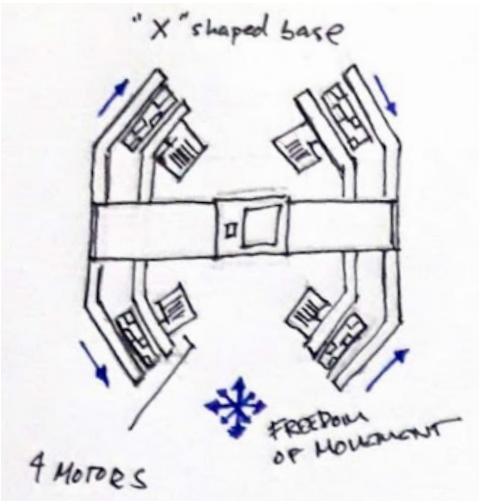
With our goals in mind, we set off to begin brainstorming ideas for building the robot. The three main components that our robot would have are: the base, the stacking mechanism, and the arm/lift.

Before choosing a design for a component, we must first analyse the pros and cons of each.

Base

Holonomic

A Holonomic base is a type of base that has its wheels and motors angled at 45°, allows for a wider range of movement through the x and y axis. It would need to use omnidirectional wheels.



Pros:

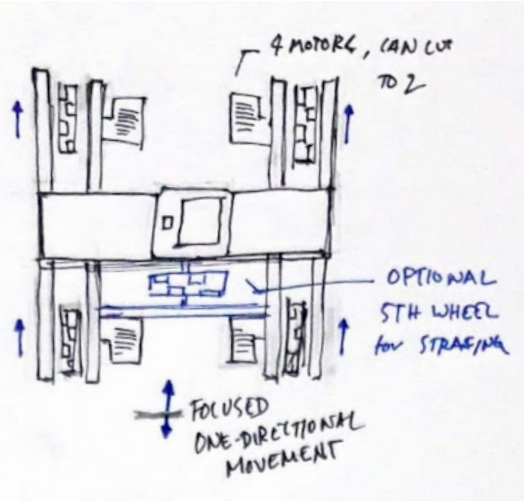
- Wider range of movement in the x and y axis
- No need to turn (useful in an autonomous)

Cons:

- Slower overall speed
- Motor power is distributed over 2 directions
- Maximum of 4 motors only

H-Base (Tank Drive)

A H-Base is a base that has its wheels and motors facing the same direction, resulting in it only limited to moving on one axis at a time. It can use either regular or omnidirectional wheels



Pros:

- Faster overall speed
- Motor power concentrated towards one direction
- Can add more than 4 motors
- Can use other types of wheels (bumpy), potentially good for climbing

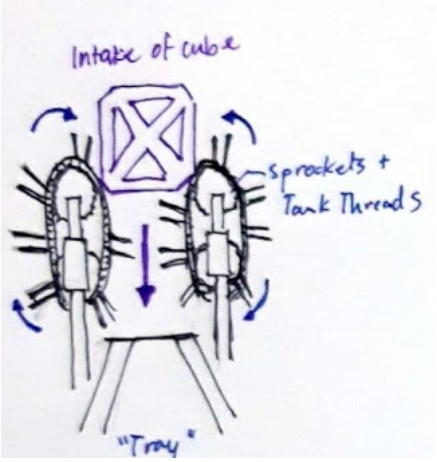
Cons:

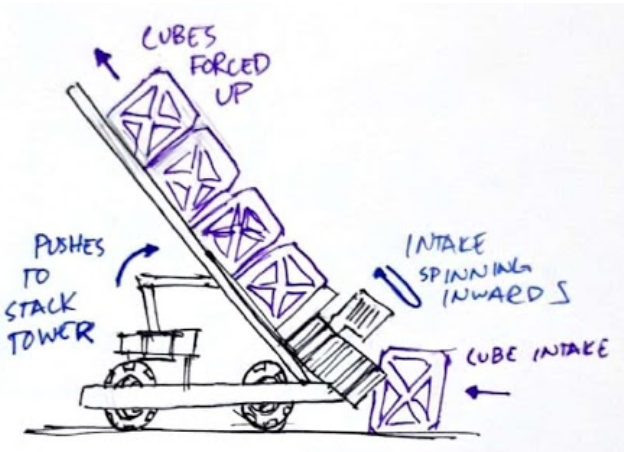
- Less maneuverability in all axes
- More difficulty turning (important in an autonomous)

Cube Stacking Mechanism

Tray Intake/Stacking Mechanism

A tray mechanism uses an intake design to collect cubes on a ramp, using gravity to keep it in place. Once enough cubes are collected, the tray can be pushed to a vertical position in order to release the cubes and score a tower in the goal zone.





Pros:

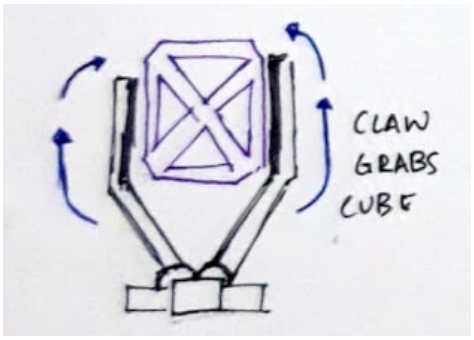
- Efficient stacking
- High cube capacity
- Relatively simple design

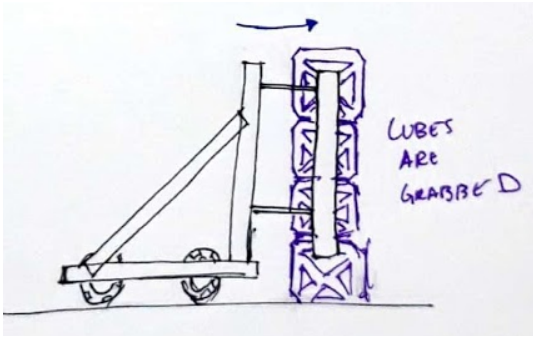
Cons:

- Minimum of 2 motors required for an intake
- Takes up a lot of space
- Cannot stack on top of existing stacks

Claw/Grabbing Mechanism

A grabbing mechanism uses a claw-like design to grasp onto two sides of a cube. It can hold this stack and create higher stacks as a result.





Pros:

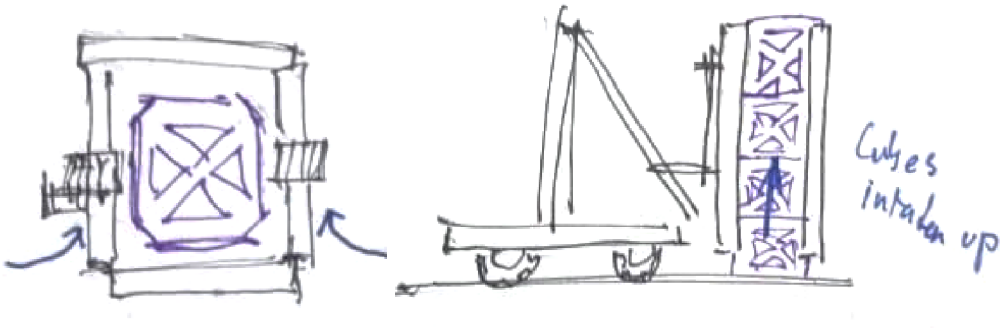
- Can use pre-set fields in the arena to stack quickly
- Can stack on top of existing stacks
- Minimum of 1 motor

Cons:

- Slower manual stacking
- Requires a lift

Passive Intake Mechanism

A passive intake uses the shape of the cube to slot into a square intake, which would collect the cube.



Pros:

- Can stack on top of existing stacks
- No motors needed

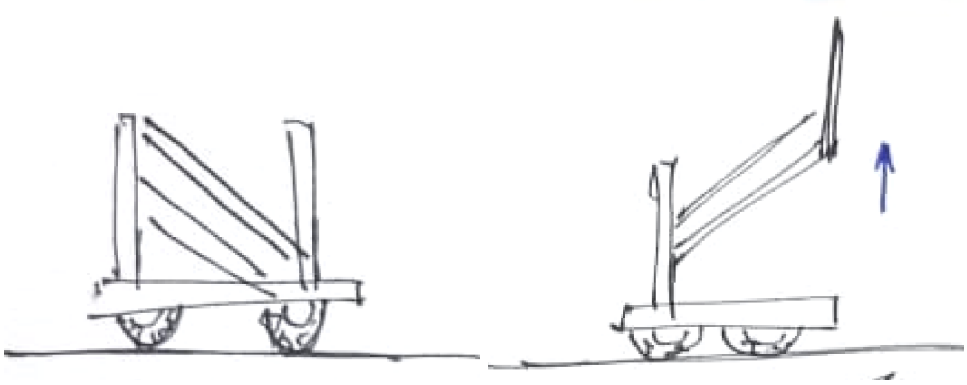
Cons:

- Slow to collect cubes
- Difficult to drive with

Tower Scoring (Extending) Mechanism

Double Reverse Four Bar

An extending mechanism which uses gears to control the angle between two c-channels, thereby controlling the length from the bottom to the top of the mechanism.



Pros:

- Easily stack blocks on towers
- Can store several blocks at once to place onto multiple towers

Cons:

- Very inefficient in stacking scattered blocks
- Takes up a lot of space
- May unbalance the robot

Simple Arm

This mechanism uses two arms to collect cubes by gripping onto them from both sides.

Pros:

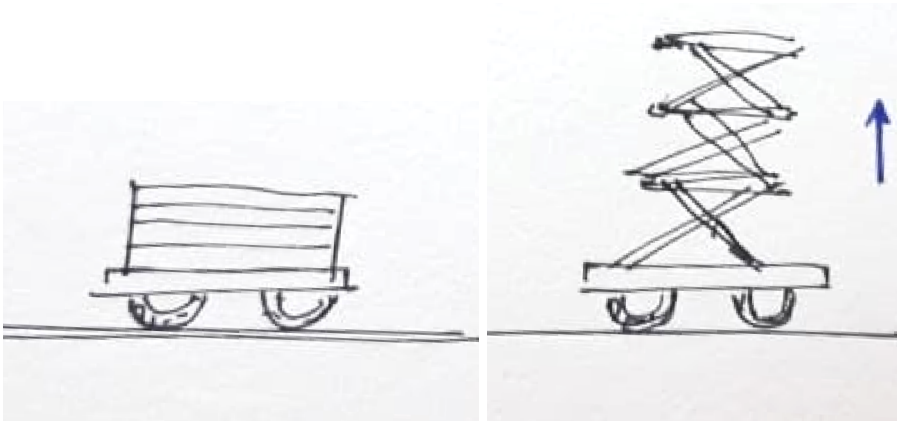
- Takes up a relatively small space
- Can also be used as protrusion to collect blocks
- Wide range of movement

Cons:

- Movement is not completely vertical, thus harder to control
- Can only grip one block at a time

Scissor Lift

An extending mechanism whose purpose is similar to that of the double reverse four bar. This uses hex screws as points of rotation to allow the metal bars to contract and expand sideways.



Pros:

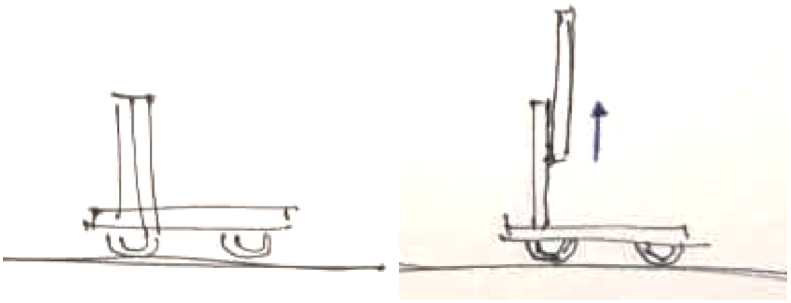
- Easily stack blocks on towers
- Can easily intake stacked blocks

Cons:

- Very inefficient in stacking scattered blocks
- Very heavy mechanism
- Takes a lot of space

Cascade Lift

An extending mechanism whose purpose is again similar to that of the scissor lift and the double four reverse bar. This uses chains to move the metal plates past each other.



Pros:

- Stable intake

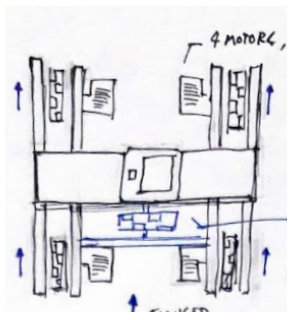
Cons:

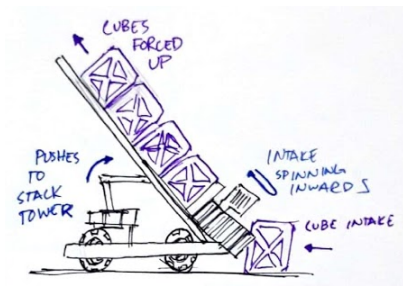
- Takes a long time to collect and deposit blocks
- Narrow/small area of support

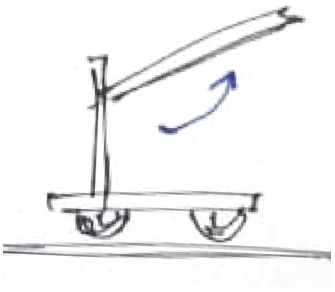
Chosen Design

Based on discussion with team members before the beginning of building for Robotics Club, we decided on the design components for each element of the robot as follows:

1. Base – **H-Base**
    - Decided due to speed and concentrated power for running through the field
2. Cube Stacking Mechanism– **Tray Intake/Stacker**
    - Decided in order to be able to easily and efficiently stack cubes in order to score in the goal zone.
3. Tower Scoring (Extending) Mechanism – **Simple Arm**
    - Chosen due to simplicity and integration with Tray intake







Why weren't other designs chosen?

Holonomic

The slower overall speed as a result of the distribution of motor power direction provided a major disadvantage. While maneuverability is important, we determined that speed towards intaking and stacking cubes is more important for scoring as much points as quickly as possible.

Claw/Grabbing Mechanism

The stacking speed of the claw mechanism averages much slower than a tray. Although preset blocks would be very quick to deposit into the goal zones, intaking blocks on the ground once the field has been scattered in a match is much slower.

Passive Intake Mechanism

Similar to the claw, the time taken to intake cubes is much slower because the cubes have to be intaken at a specific angle, going upwards. Further, it is much harder to deposit stacks because the whole intake has to be lifted out of the tower.

Double Reverse Four Bar

We've determined that we didn't necessarily need the added complexity and benefits of the DR4B. Because it takes up a lot of space, we decided to use that space to build up simple stacking capabilities with the simple arm. We also did not want to risk tipping the robot over while extending.

Scissor Lift

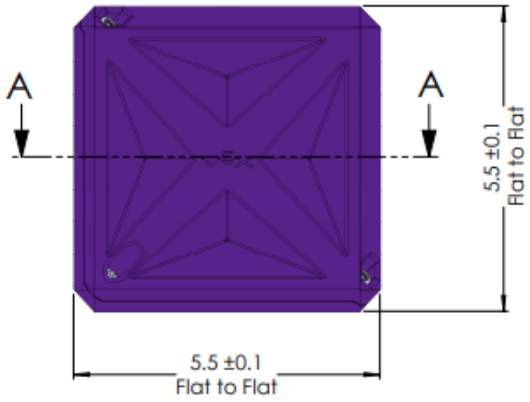
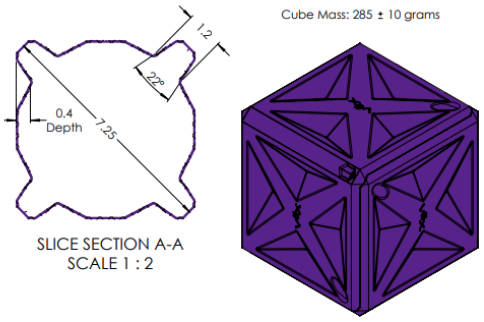
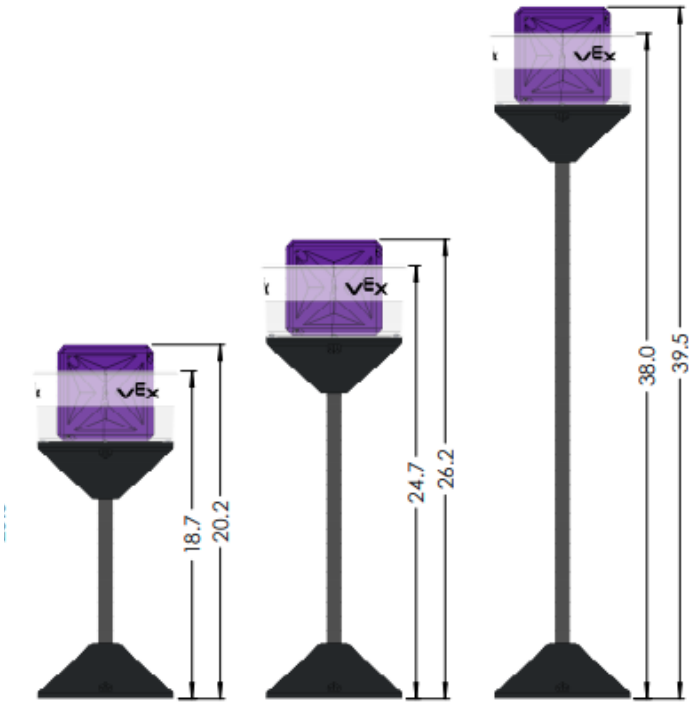
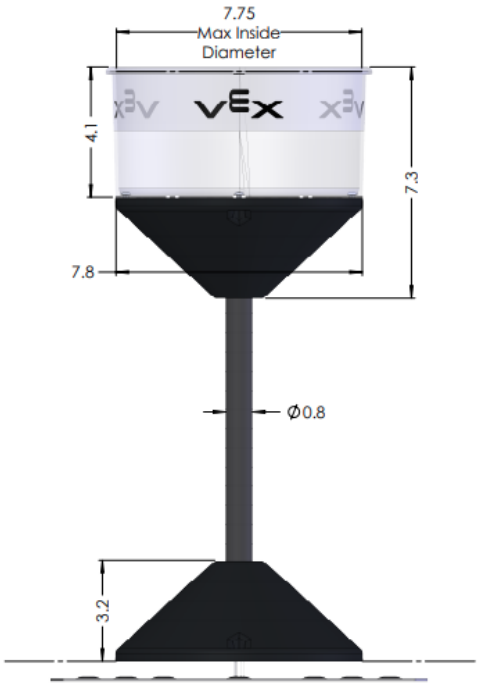
The scissor lift is generally much slower in extending up because of the amount of points of contact. This creates more friction and thus more force needed to be exerted on the heavy structure. Further, the lift takes up a lot of space that can't be utilised.

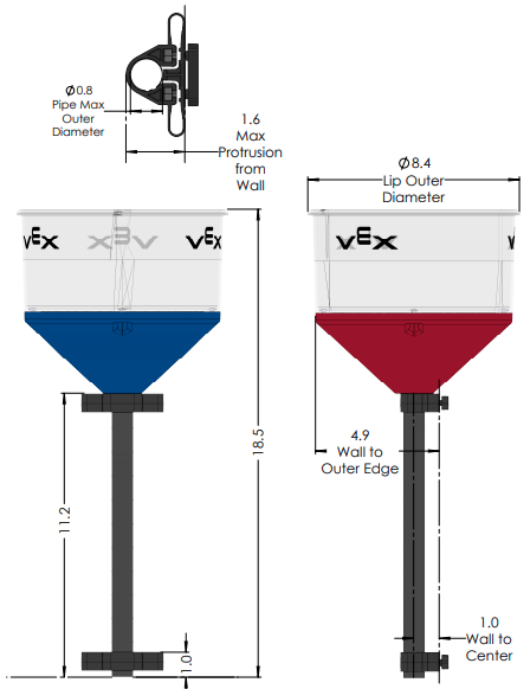
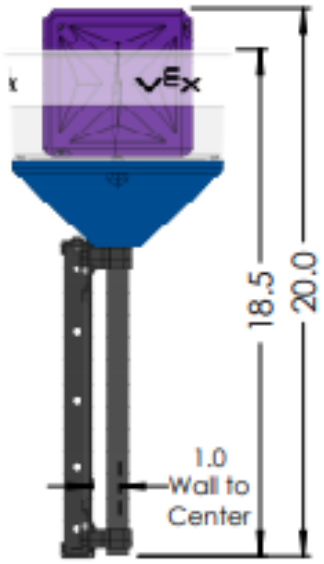
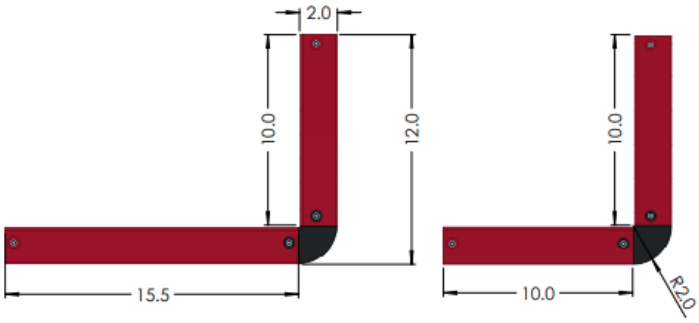
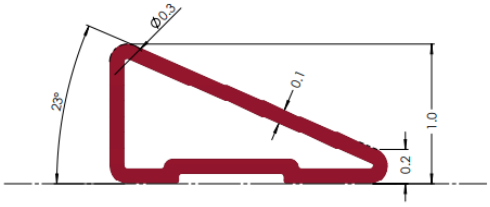
Cascade Lift

The cascade lift is much slower, and similar to the scissor lift, has a lot of points of contact and friction which would require more power to be exerted.

With these in mind, we began designing a rough draft of what the robot would look like, more specifically the main base in order to get straight into building as soon as Robotics Club starts.

Game Kit Specifications

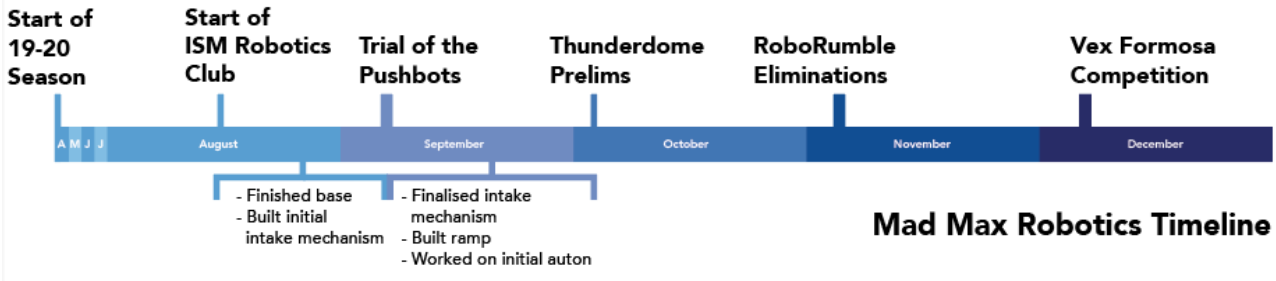
Cube Specs		
Tower Goal Specs		

Alliance Tower Goal Specs		
Goal Zone Specs		

Source:
https://content.vexrobotics.com/docs/vrc-tower-takeover/TT-AppendixA-20190816.pdf

Key Dates for the Competition

Date	Event	Description	Notes
8/19/19	Start of Robotics Club	First day of school Robotics Club. Introduction of competition, and beginning to find parts for construction.	Building was delayed as the club had to learn how to use V5 for the first time
9/9/19	Trial of the Pushbots	First elimination match for teams. Requirement for teams is they must have a robot with a working base that can move.	1 week between matches and skills. Placed 1st overall, 2nd in matches and 1st in skills
10/4/19	Thunderdome Prelims	Second elimination match for teams. Played in the style of the competition. The top 8 teams make it to RoboRumble.	Placed first overall for the regular matches and for driver skills, winning by a large margin. Need to work on the consistency of autonomous skills.
11/8/19	RoboRumble 2019	Final elimination match for ISM robotics teams, qualifying match for the Philippines. Played in the style of the competition, complete with skills challenges. Top 3 teams get to compete in Taipei.	
12/6/19	Taipei Vex Formosa Competition	Vex International Competition for Asia-Pacific Region. Played in the style of the competition, complete with skills challenges. Top teams make it to Vex Worlds.	



Project Timeline

Deadline	Target	Description
August 27, 2019	Basic V5 Coding	By this date, all of our team members already completed the 4 basic programming challenges, which included basketball drills, the maze challenge, the square challenge, and the competition template. This served as a prerequisite to all the robotics club members before they could start on the physical robot.
Before September	Robot base	We ensured that the robot's base was firm, stable, and straight to maintain precision.
Before September	Basic movement programming	We created a basic program to allow us to drive the robot using the joystick.
2nd week of September (September 14, 2019)	Robot arms and intake mechanism first prototype	We attached the arms and a rudimentary system to be able to rotate the intake mechanism.
2nd week of September (September 14, 2019)	Ramp attachment	We attached a ramp to the robot, as well as the gear mechanism which enabled us to control the ramp's degree of sloping.
3rd week of September (September 21, 2019)	Autonomous skills program	We created a basic program which directed the robot to score in the goal zones.
3rd week of September (September 21, 2019)	Ramp pushing mechanism	This mechanism consisted of gears which pushed a group of c-channels forwards or backwards, thus enabling us to control the ramp's degree of sloping.
4th week of September (September 28, 2019)	2nd layer of ramp/tray	This 2nd section of the ramp was added and designed to flip outwards caused by the intake of blocks.

4th week of September (September 28, 2019)	Intake mechanism folding	The intake mechanism initially didn't fit within the bounds, so a "rubber band vertex" system was used to be able to firmly grip the blocks while extended and stay within bounds while folded. This required a lot of testing.
3rd week of October (October 19, 2019)	3rd layer of ramp/tray	The third layer of the ramp proved to be a very difficult challenge to overcome. A large amount of testing and prototyping was done to create an automatically extending 3rd section of the ramp.

JOURNAL ENTRIES

All journal entries written with the date and person who wrote it in parenthesis. Each entry begins with a summary of the date, and bolded portions highlight key points. At the end, the front, side, and top views of the robot are taken from that day (when available).

August 19, 2019 (John)

Summary:

For our club's first session, all teams were required to accomplish a set of programming challenges before they can start building in order to transition from RobotC to vexcode. Pre-made robots were provided. However, I took this time to create and test the following functions that should prove to be handy in the future:

```
double todeg(double target) {
  double wheelDiameter = 10.16;
  double circumference = wheelDiameter * 3.141592;
  double degreesToRotate = 360 * target/circumference;
  return degreesToRotate;
}
void forward(double distance, double velocity) {
  double deg = todeg(distance);
  LeftMotor.setVelocity(30,vex::percentUnits::pct);
  RightMotor.setVelocity(30,vex::percentUnits::pct);

  LeftMotor.rotateFor(deg, vex::rotationUnits::deg,false);
  RightMotor.rotateFor(deg, vex::rotationUnits::deg);
}
void turn(bool direction, double degrees, double velocity) {
  LeftMotor.setVelocity(velocity, vex::percentUnits::pct);
  RightMotor.setVelocity(velocity, vex::percentUnits::pct);
    if(direction == 1) {  //left
    LeftMotor.rotateFor(-1 *degrees,vex::rotationUnits::deg,false);
    RightMotor.rotateFor(degrees,vex::rotationUnits::deg);
  } else if(direction == 0) {//right
    LeftMotor.rotateFor(degrees,vex::rotationUnits::deg,false);
    RightMotor.rotateFor(-1 * degrees,vex::rotationUnits::deg);
  }
}
 //deprecated. Only does quarter or 90° turns.
void qturn(bool direction, double velocity) {
  LeftMotor.setVelocity(velocity, vex::percentUnits::pct);
  RightMotor.setVelocity(velocity, vex::percentUnits::pct);
  //left
  if(direction == 1) {
    LeftMotor.rotateFor(-240,vex::rotationUnits::deg,false);
    RightMotor.rotateFor(240,vex::rotationUnits::deg);
  } else if(direction == 0) {
    //right
    LeftMotor.rotateFor(240,vex::rotationUnits::deg,false);
    RightMotor.rotateFor(-240,vex::rotationUnits::deg);
  }
}
```

These functions are incorporated to create a shorter and more efficient solution to the following problems. Note that the test robot has 2 motors and thus the code will be changed based on the structure of our robot's drivetrain in the future.

1. **Basketball Drills** - make a robot go forwards 30cm, backwards 30cm, forwards 90cm, and backwards 90cm.

```
void basketball() {
forward(30,20);
vex::task::sleep(1000);
forward(-30,20);
vex::task::sleep(1000);
forward(90,20);
vex::task::sleep(1000);
forward(-90,20);
}
```

2. **Maze Challenge** - make a robot reach the end of a set "maze".

```
void maze() {
 forward(60, 30);
 vex::task::sleep(1000);
 qturn(1,30);
 vex::task::sleep(1000);
 forward(55,30);
 vex::task::sleep(1000);
 qturn(0,30);
 vex::task::sleep(1000);
 forward(55,30);
 vex::task::sleep(1000);
 turn(0,320,30);
 vex::task::sleep(1000);
 forward(70,30);
}
```

3. **Square Challenge** - make a robot go around a block in a square path. Students must use a for loop to accomplish this challenge.

```
void square() {
 for(int i = 0; i < 4; i++) {
  forward(100,30);
  vex::task::sleep(500);
  turn(1,270,30);
  vex::task::sleep(500);
 }
```

4. **Competition Template Challenge** - students must be able to run code using the competition template.
    a. For this challenge, any code that shows distinguishable differences between the autonomous mode and driver control mode would pass. We simply put one of the code solutions in the usercontrol() function to pass.

All future codes will follow competition standards and a full version will be present in the appendix.

August 27, 2019 (Justin)

Summary:

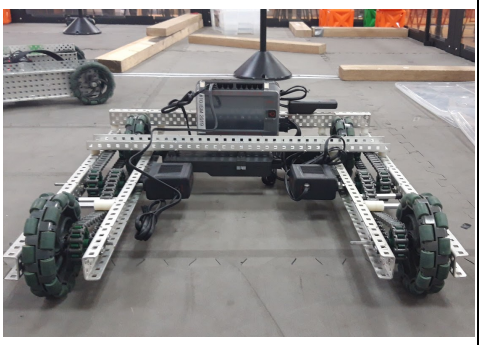
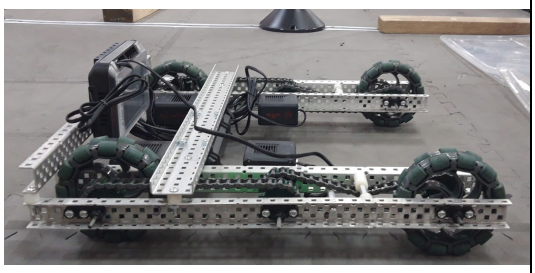
1. Completing the base

During this session of robotics we primarily focused on completing the base of the robot. We placed the brain of the robot facing the inside of the base in order to fit within the size specification. We also did this because there should be enough space for the wiring to fit inside the gap left where our lift will be. Our team worked relatively divided during this session, with some focusing on building and the others finishing the coding challenges, so as to allow everyone to be able to build and work on the robot.

Coding (John):

The code here implements a basic tank-style control for the robot with the use of a remote. It is based on two joysticks, with each one's vertical axis controlling the corresponding side of the robot. It is programmed such that the joystick's position is regarded as the percentage velocity the wheels will turn. The tank control gives the user the most control over the robot's movement. From here on the competition template is used for programming the robot.

```
void tank() {
  LeftFront.spin(vex::directionType::fwd, controller1.Axis3.position(), vex::percentUnits::pct);
  LeftBack.spin(vex::directionType::fwd, controller1.Axis3.position(), vex::percentUnits::pct);
  RightFront.spin(vex::directionType::fwd, controller1.Axis2.position(), vex::percentUnits::pct);
  RightBack.spin(vex::directionType::fwd, controller1.Axis2.position(), vex::percentUnits::pct);
}
```

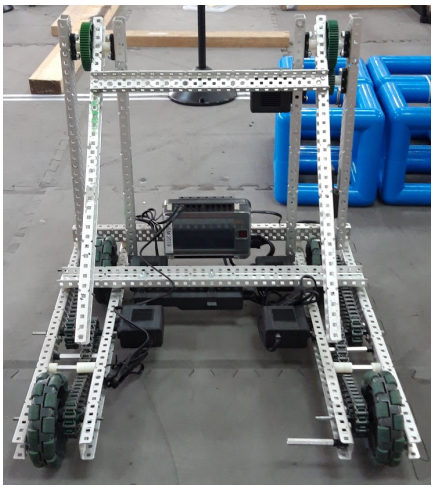

The completed base, with the brain and battery attached, and the motors wired

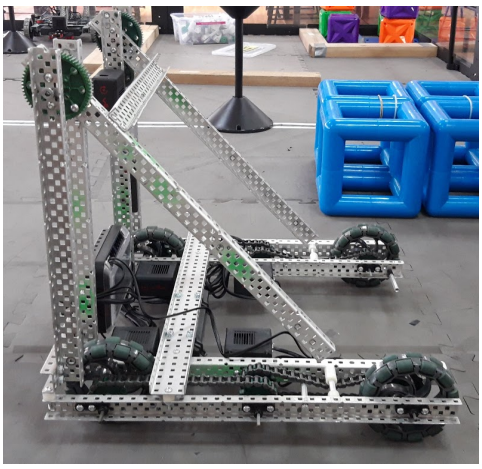
August 31, 2019 (Eion)

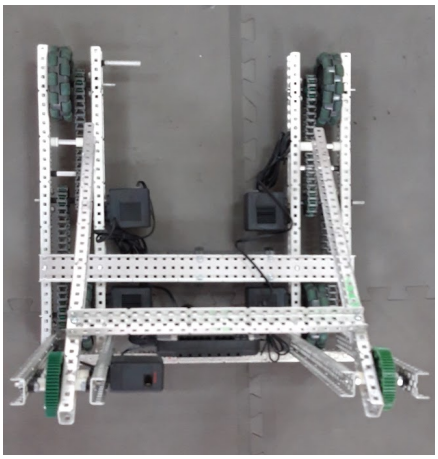
Summary:

1. Beginning the construction of the arm

Today we began to construct the robot's arms. In order to do this, we first set up vertical poles to which the arms will be connected to and supported by. These arms were set to be slightly shorter than the **base's length (18")** when suspended diagonally. The intake mechanism would later to be attached to these arms to pull in blocks. Ending just behind the wheels, the arms allow the intake mechanism to reach further than the robot's base to pick up cubes more effectively; as well as still providing more space for the arms to be folded in within the size limit. **The robot's arms are essential to scoring, as these will be both the intake system used to both pull blocks up the ramp for stacking blocks on goals, as well as the gripping system used to carry the blocks to place them on towers.** A motor was connected to a small gear, which in turn was placed adjacent to the large green gear, which can be seen in the pictures. This creates a low gear ratio, which ensures that the cubes will be lifted slowly, and thus, keeps them stable. An important thing to note, however, is that this year, we will only be able to use a maximum of **8 motors** for the robot. It is likely that 2 of these motors will be allotted towards the arm-lifting mechanism.





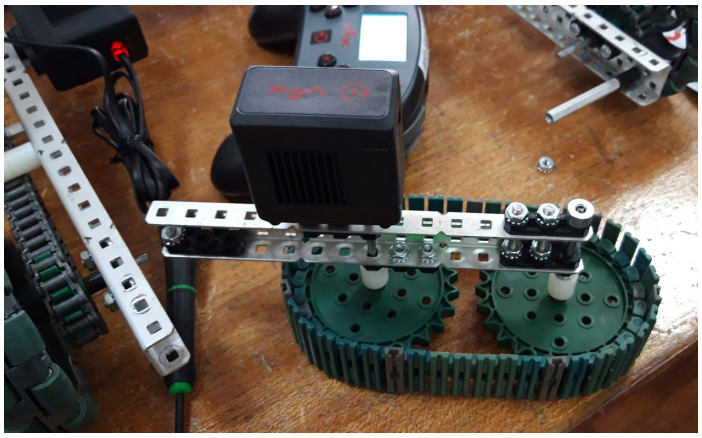


The robot's base frame, with rudimentary arms attached.

September 2, 2019 (Seba)

Summary:

1. Creating the design for the roller intake
2. Testing the roller intake design

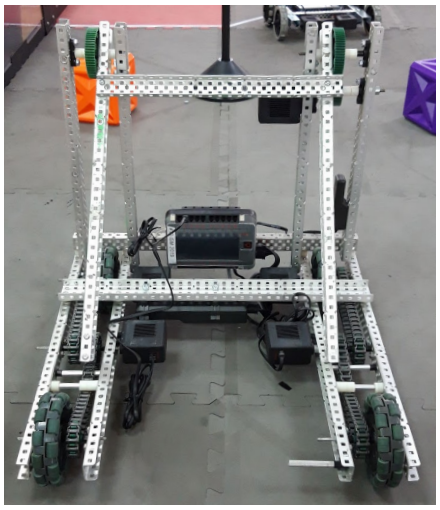


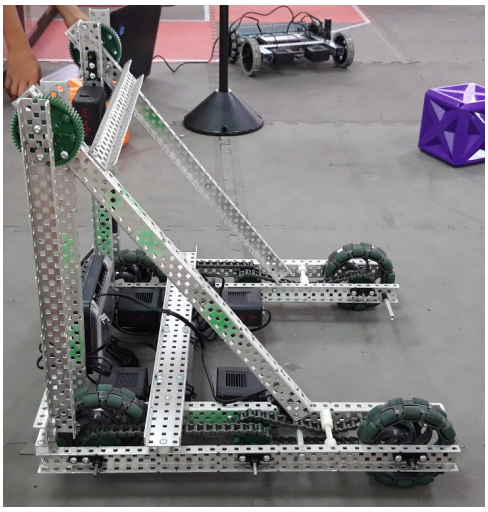
During this session we built a prototype design for the cube intake of our robot. Both segments of the ball intake had a motor attached to it so that it could generate enough power to lift the cubes. We are still working on choosing a way to keep the balls from falling out of the intake when trying to lift the cubes. We also began planning ways to make the intakes of the robot retract outwards. We also began to finalize the design of our arm, with the distance it currently being at being our final arm length for the robot.

The majority of our team had finished the coding challenges at this point and had moved onto working on the robot. As such, work on the robot began to move at a faster pace, though with some roadblocks. The primary challenge we had come across in the time was organizational, with a number of tests being scheduled during this week and the following week after.

Coding:

During the creation of the intake segment of the robot, we also created a series of test codes in order to see if the intake worked. This was met with some difficulty as the intake would spin one direction, but have difficulty spinning the other way. This ultimately became the primary activity of the day: making sure the intake was working properly.



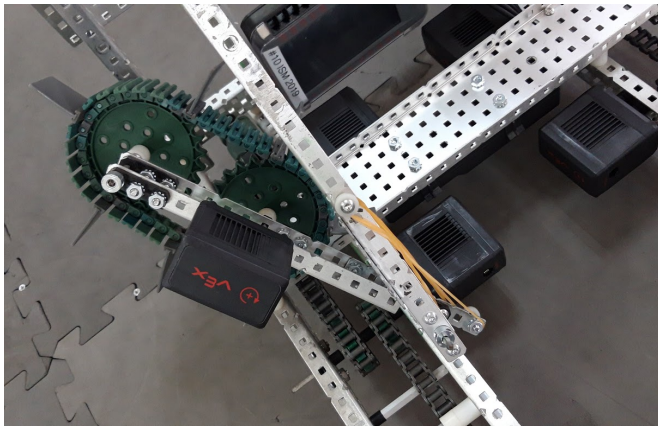


Photos of the robot as of September 2, 2019

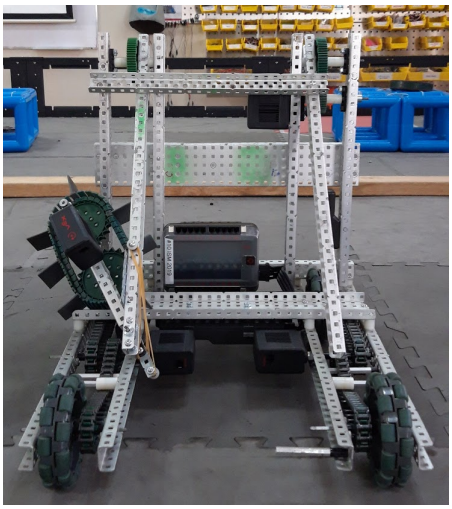
September 6, 2019 (Seba)

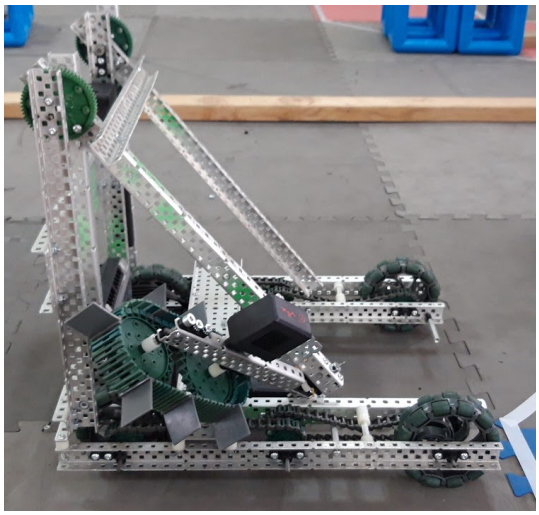
Summary

1. Creating the retracting mechanism for the roller intake
2. Testing the retracting mechanisms



The majority of the time spent on this session was focused on the creation of a retracting method for the roller intake. During our research on robots with a similar design to ours, we noticed that people were using a locking mechanism which would lock the roller intake in place once it reached its desired point. This proved to be very challenging to recreate and we ultimately decided to use trial and error on a series of different designs that used rubber bands keep the roller intake in place. At the end of the session, we came to the issue that we needed positions the rubber bands somewhere on the robot where it could keep the roller intake inside the size specification.



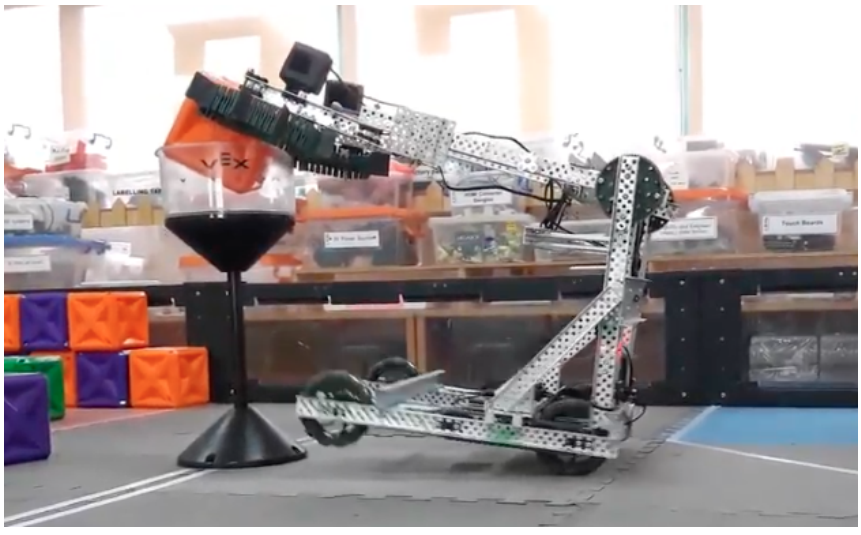


Photos of the robot with the trial version of the retracting intake

September 10, 2019 (Justin)

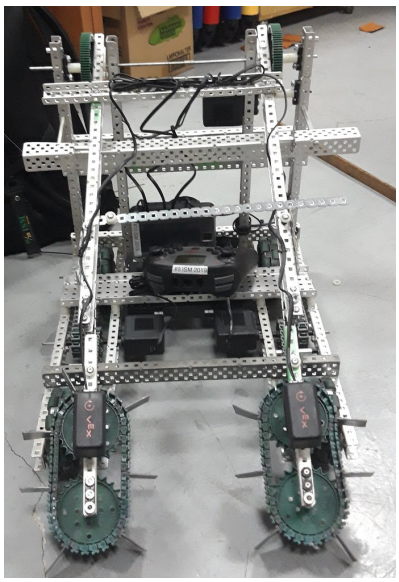
Summary

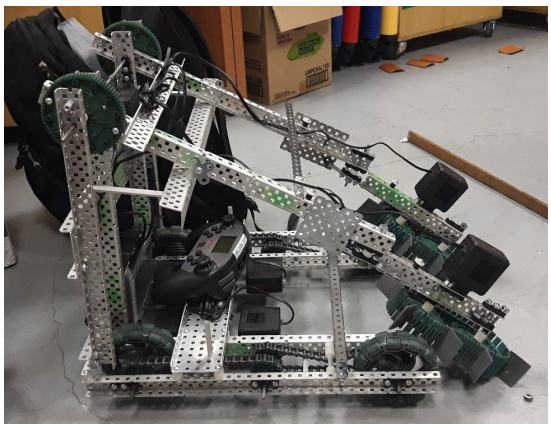
1. Attached both rollers
2. Connected and reinforced arm
3. Tested goal post scoring

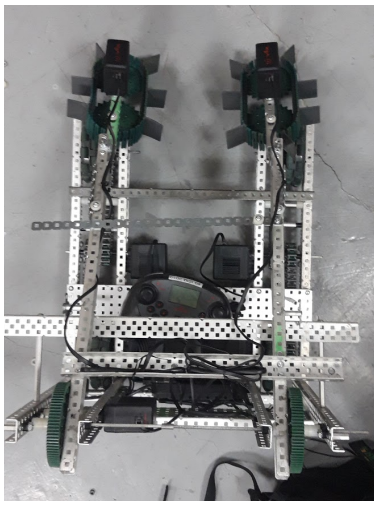


On this session we started testing if the robot was able to place a cube onto one of the towers. This task proved successful, albeit with difficulty. We had noticed that on occasions, the robot would drop the cube it was holding because it did not exert very much pressure onto the cube. We came to the solution of making the arms bend

inwards by using a bar that holds it in a triangular shape. This proved to be successful, but needed modification. We also spent time reinforcing the arm of the robot. The arm had the issue of one side lifting at a faster rate than the other. This was due to there only being one motor attached to the arm on one side (see the photos). To remedy this, we ran an axel from the motor to the other side of the robot. This allowed the arm to lift at s more even rate, though we need to improve this later.





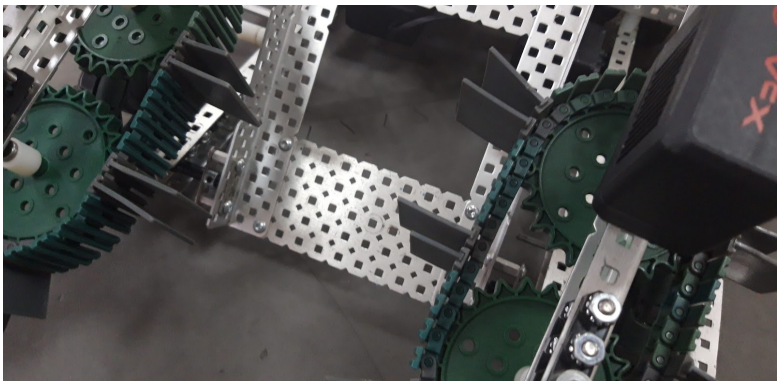


Photos of the robot as of September 10, 2019

September 14, 2019 (Eion)

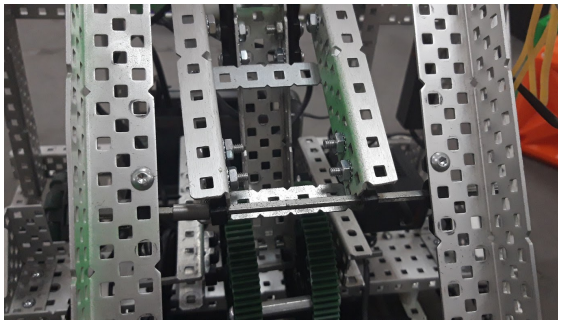
Summary:

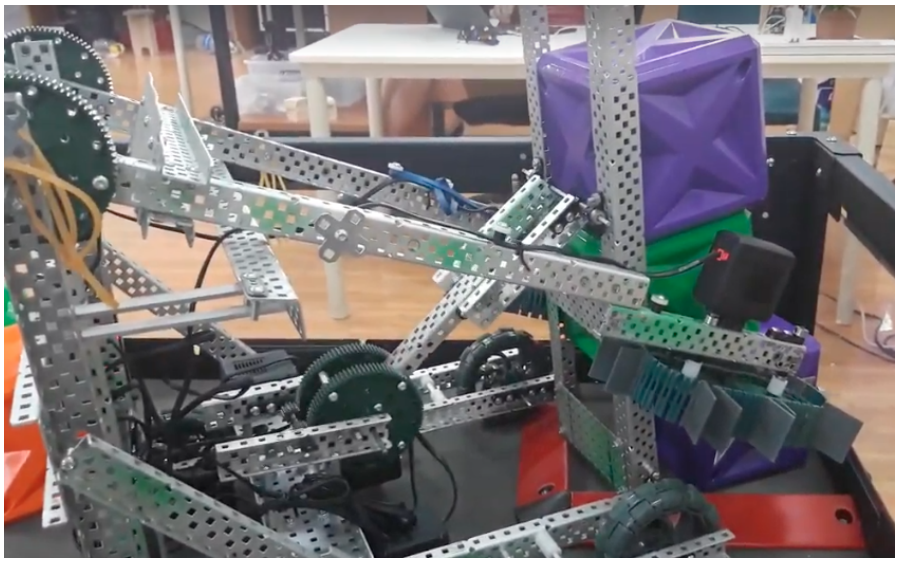
1. Worked on intake mechanism
2. Attached ramp/tray for loading cubes

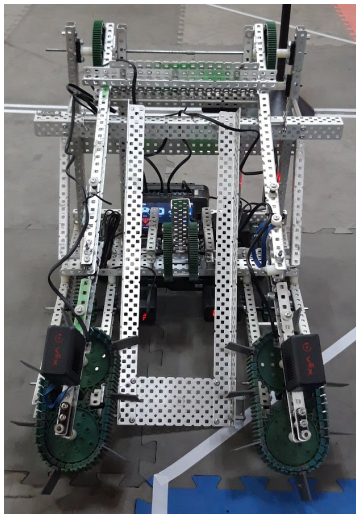


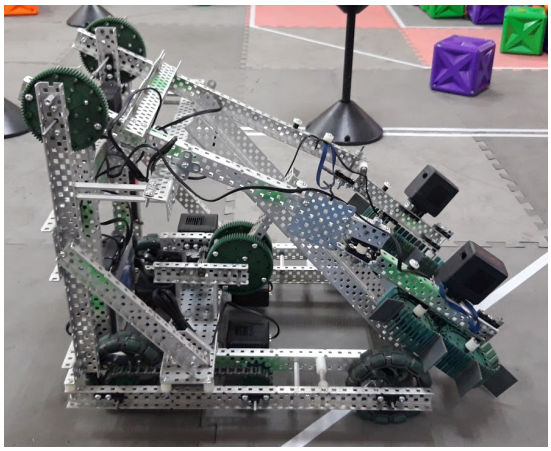
The intake mechanism here refers to the gears, treads, and rubber sheets, as well as the metal c-channel to which all those parts are attached, that pulls the blocks quickly onto the loading mechanism. In

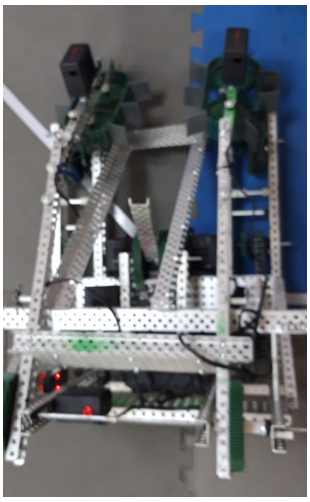
the beginning of the process, the motors attached to the arms start rotating the gear directly connected to it, which in turn rotates the treads. The other wheel's purpose is basically to increase the inward-facing surface area of the gripping mechanism i**n order to more efficiently and consistently pull the blocks inwards and upwards**. The rubber sheets are very important too in this process, as these are sufficiently firm while being pliable as well. Thus, while moving along relative to the mechanism at high speeds, these **rubber sheets can easily pull in blocks while not obstructing the passage of other blocks further along the path**. In addition to that, without the mechanism rotating, the rubber sheets also prevent the cubes from falling back down onto the ground and thus keep them on the ramp, which leads us on to the next major component of the robot. The ramp works in direct conjunction with the intake mechanism, since it is this that contains the blocks that are pulled in. We measured the length of the cube's edges and thus fit the width of the ramp to be that size as well. **As of this iteration, the robot can hold up to 3 blocks at once, as seen on the right.**











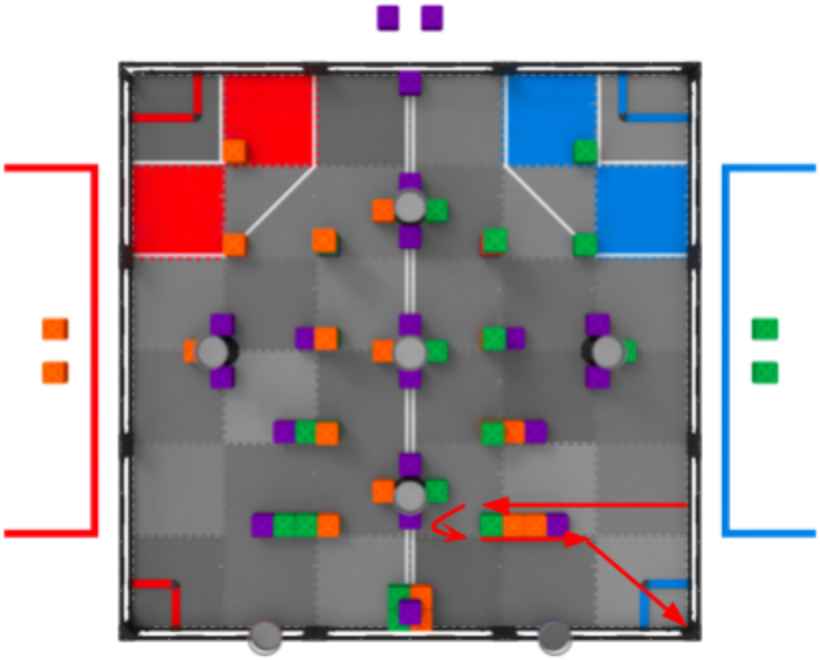
Front, Side, and Top views respectively of the robot with both the intake mechanisms (arms) and loading ramp attached.

September 17, 2019 (John)

Summary

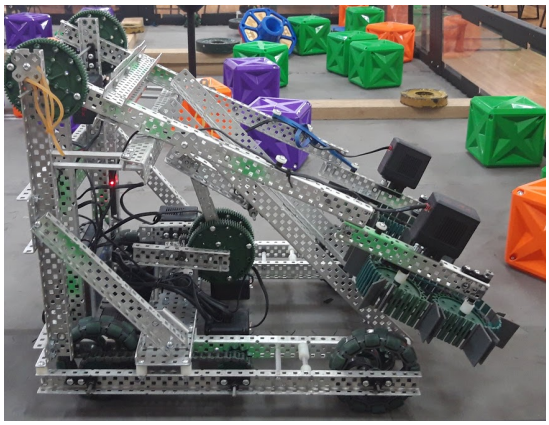
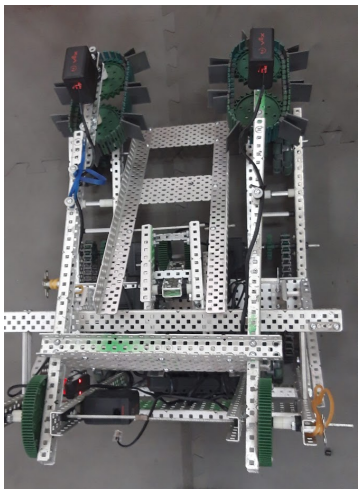
1. Skills Autonomous Code(3pt)

There are two types of skills challenges: driver and autonomous. In each of them, the robot is given 1 minute to display its ability. For the first test of autonomous code, the robot was able to stack three blocks in a scoring zone. The limitation was mainly due to the intake mechanism of the robot not having enough space for four blocks. However, it was able to stack three blocks very reliably. The path of the robot is shown in the image.



```
void autonomous( void ) {
//Setting Velocity
LeftFront.setVelocity(35,vex::percentUnits::pct);
LeftBack.setVelocity(35,vex::percentUnits::pct);
RightFront.setVelocity(35,vex::percentUnits::pct);
RightBack.setVelocity(35,vex::percentUnits::pct);
armLeft.setVelocity(100,vex::percentUnits::pct);
armRight.setVelocity(100,vex::percentUnits::pct);
//Start spinning arms to intake blocks
armLeft.spin(vex::directionType::fwd);
armRight.spin(vex::directionType::fwd);
//Go forward 70cmGo forward 70cm
forward(70);
armLeft.stop();
armRight.stop();
vex::task::sleep(500);
//Make arms stop intaking, but still spin slowly to keep the cubes held
armLeft.setVelocity(10,vex::percentUnits::pct);
armRight.setVelocity(10,vex::percentUnits::pct);
armLeft.spin(vex::directionType::fwd);
armRight.spin(vex::directionType::fwd);
//Go backward 45cm
forward(-45);
vex::task::sleep(500);
//Make the robot turn towards the scoring zone
turn(1,300,15);
vex::task::sleep(500);
```

```
//Make the robot move (30cm) towards the scoring zone
forward(30);
//Make the angle move to make the blocks perpendicular to the ground
angle.setVelocity(25,vex::percentUnits::pct);
angle.rotateFor(1000,vex::rotationUnits::deg,false);
//Make the arms spin in reverse to release the blocks
armLeft.setVelocity(-30,vex::percentUnits::pct);
armRight.setVelocity(-30,vex::percentUnits::pct);
armLeft.spin(vex::directionType::fwd);
armRight.spin(vex::directionType::fwd);
vex::task::sleep(4000);
//Make the robot move backwards after scoring the blocks
forward(-50);
armLeft.stop();
armRight.stop();
}
```

The robot's design during the first autonomous skills challenge

September 9 & 18, 2019: Trial of the Pushbots (Jason)

Summary:

1. 1st place overall in scrimmage
    a. 2nd in matches, 1st in skills

Trial of the Pushbots is **ISM's first elimination round for teams.** The only basic requirement is to have a driveable base working with the competition template code. Other than this, it is a great first experience of getting used to Tower Takeover's competition format, in both Matches and Skills. In essence, it is a scrimmage.

Our **main goal was to get accustomed to the competition**, and see how both we and the robot are able to work both on and off the field. The first day were the matches. We placed 2nd, where we won 1 of our matches, having a 1 pt scoring auton by pushing a cube into the goal zone, and tied in the other, as all teams scored 2 pt autons and 3 pts for cubes. We had a high score of 3 pts during the driver control period, the total number of cubes that can be placed on goal zones.

The second day, which happened a week after matches, consisted of skills, both programming and driver. I drove for the driver attempts, while John tested his current autonomous. *(This was originally supposed to be on Aug. 11, but got pushed back due to typhoons) We continually refined the code to the point where it could score **3 pts in auton** by stacking 2 cubes and a preload. We placed 1st, scoring 3 pts in auton and **11 pts in driver control**– the best score within the club at that point. I did so by stacking 2 stacks of 4 cubes, and another stack of 3. **Overall, we placed 1st.**

Our first experience with the competition showed us our strengths and weaknesses, and gave us the opportunity to adjust and improve. It was decided that I will be the main driver for the robot until we potentially implement dual controllers in the future. Apart from this, we pinpointed flaws and weaknesses of the robot, including the state of the cap flipper, climbing, as well as in terms of coding.

TEAM ANALYSIS #1

Trial of the Pushbots was also a **good opportunity to scope the competition** and keep track of any notable teams. There are 10 total teams competing in Trial of the Pushbots, one of which was eliminated (as it was disbanded). In order to get a sense of the competition for the Manila/ISM season, we did some analysis on how each team performed based on their results, past experience, and potential functions of their robot, given as to how detrimental scoring strategies are in this year's VRC. We understood, however, that these rankings are tentative and may be subject to change as each team progresses.

At the moment, **we are the only ISM team with stacking capabilities,** with a maximum of 4 stacks at the present. This allowed us to achieve the best score so far for driver and programming skills. However, we expect teams to continue to develop as the season progresses.

Most teams aimed at being able to push a cube into their zone. As of now, all ISM teams have opted for and are working towards a "complex tray" design.

- **Johnny Boi**, a mixed grade team, placed 2nd overall, with 1st in matches and 5th in skills. At the moment, they have a pushbot to score cubes in their goal zones.
- **NAIA na lang** was also a notable and surprising team, as they were all freshmen. Once again, they also built a pushbot capable of pushing blocks into their goals. They scored the 2nd highest in skills, suggesting good driving.
- **Euphoria**, another mixed grade team, placed 4th overall. They placed 3rd in skills, able to score at least 1 cube in all goal zones. They were successful with managing their driving during skills.
- **C5 Extension**, another all freshmen team, placed 5th. They had a good auton, placing 3rd in matches with 9 AP.
- **Athena** and **Munchlax**, both all junior teams, placed 6th and 7th respectively, while also tied in points. Athena was one of the only teams to score in programming. We expect them to be more experienced than younger teams.
- **EDSA**, an all sophomore team who went to Worlds for Turning Point, underperformed and placed 8th. We expect to see them develop more into the season, and as such, we will need an eye out for them.

- **Cream**, an all senior team, also underperformed, placing 9th. However, they may have a greater amount of experience compared to freshmen, sophomore, and junior teams.

Below is the full list of the results from Trial of the Pushbots. Things of note include tie breakers, Autonomous Points & Programming High Scores, among others. Note: The top 8 teams from Thunderdome advance to RoboRumble.

Trial of the Pushbots Results												Notes:
No.	**Team**	**Matches**				**Skills**				**Overall**		The total of the rank in Matches and Skills is used to determine the overall score. The lower the score, the higher the rank.
		WP	AP	SP	**Match Rank**	P HS	D HS	Total	**Skills Rank**	**Score**	**Rank**	
1	3818 A Mad Max	3	9	9	2	3	11	14	1	3	**1**	
4	3818 D Johnny Boi	4	6	4	1	1	2	3	5	6	**2**	In case of a tie, Skills Rank is the first tie breaker.
8	3818 H NAIA na Lang	2	0	4	6	1	5	6	2	8	**3**	The tie between NAIA na lang and Euphoria was broken by skills
7	3818 G Euphoria	2	6	1	5	0	4	4	3	8	**4**	
10	3818 J C5 Extension	3	9	6	3	0	2	2	8	11	**5**	
5	3818 E Athena	2	0	4	7	1	2	3	5	12	**6**	The tie between Athena and Munchlax was broken by skills
6	3818 F Munchlax	3	3	7	4	0	2	2	8	12	**7**	
2	3818 B EDSA	1	3	7	8	0	3	3	6	14	**8**	
3	3818 C Cream	0	0	1	10	0	2	2	8	18	**9**	
9	3818 I LiPo Tarsier	0	0	1	10	0	0	0	10	20	**10**	LiPo Tarsier disbanded during the event

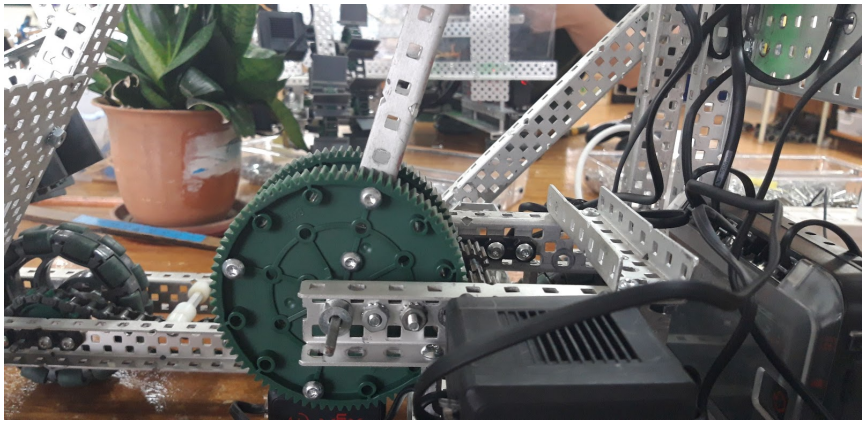
Post scrimmage reflection points and improvements:

1. Improvements for stacking consistency
    a. Controls for the process of stacking are incredibly finicky. Jason needs significant practice and improvements to the code must be implemented (ideally would be a code that can push the tray and release the stack in one press of a button)
2. Formulate a plan of action for driver and auton skills; consider the most efficient pathway in terms of distance and point gained
    a. Going towards the smaller goal side, with the 4 flat cubes on the floor, proved to be an effective place to easily get cubes to stack
3. Fix the inability of the robot to lift its arm and score on high posts. At the moment, the arm was not functional and could not lift both sides simultaneously
4. Minimize the footprint of the robot to have it fit in the 18” x 18” x 18” size limit. This includes the roller intakes and tray.

September 20, 2019 (Jason)

Summary:

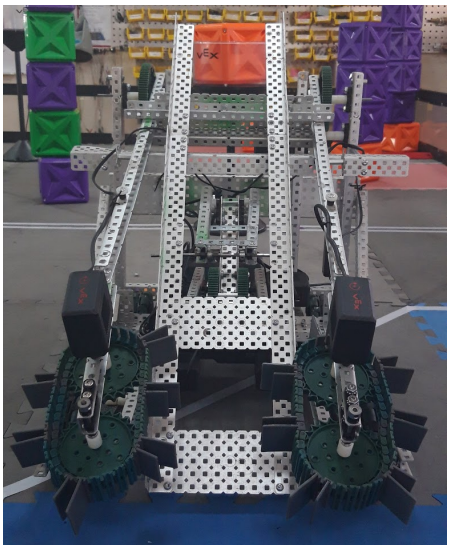
1. Fixing the tray pushing mechanism

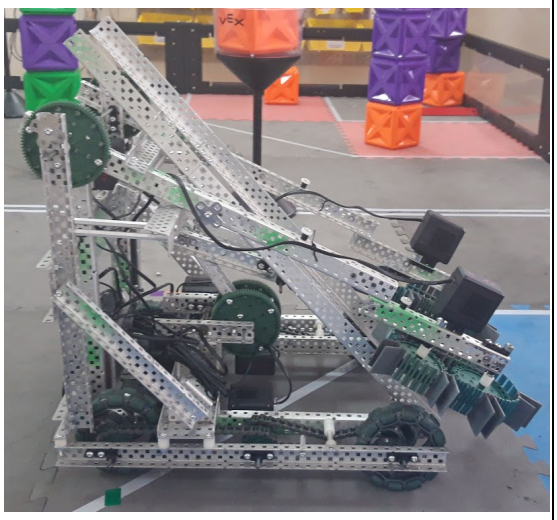


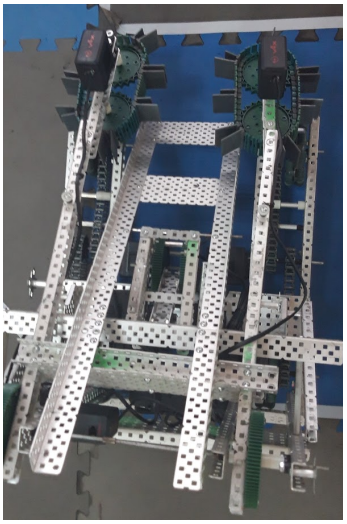
The primary focus of today was on **fixing the tray pushing mechanism**. After Trial of the Pushbots, it was observed that the positioning and installation of our **tray became flimsy and looser than normal**. The standoffs that we used to raise the mechanism consisted of two 1.5” pieces joined with a connector screw. The vibrations experienced while driving and using the tray loosened these screws, and thus prompted us to replace them. We used 3” standoff pieces to eliminate any excess failure points when lifting the mechanism above the

base. The mechanism itself was firmly reattached in a new, central position in order to increase stability when stacking.

Further, the c-channels used in the joint section of the tray mechanism were **reassembled** with delrin bearings and axle connectors. This new positioning allowed for a more vertical angle for the tray when placing the stacked blocks in a goal zone. Rebuilding this section further increased the stability of this section, which made intaking much smoother than previously.





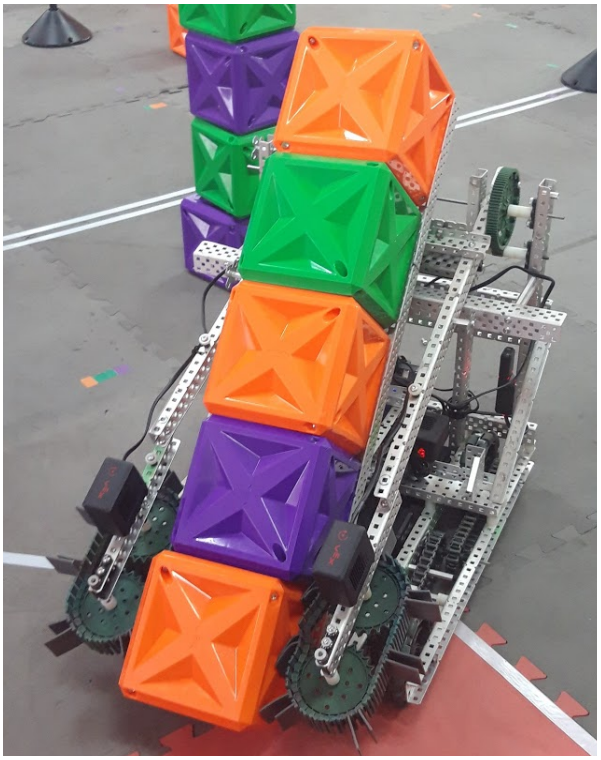


The robot with the newly renovated tray extension mechanism.

September 21, 2019 (Jason)

Summary:

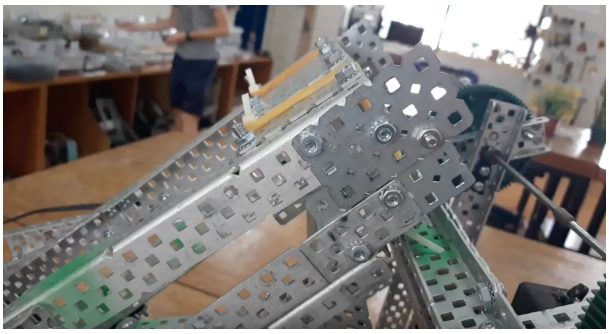
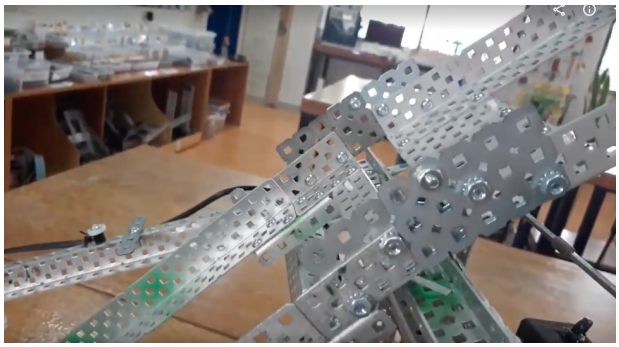
1. Experimenting with the tray extension
2. Practice Driving

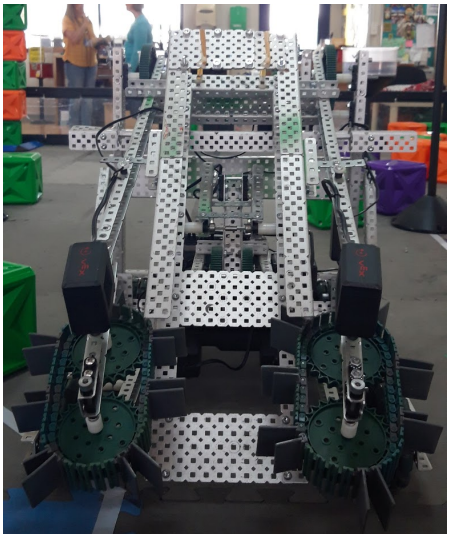


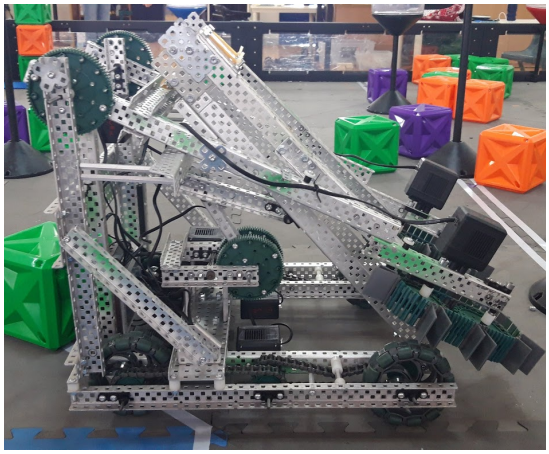
Today I attempted to do some **practice driving with the robot at its current state**. During Trial of the Pushbots, our maximum capacity was ~3 blocks because of the rushed nature of its setup for doing skills runs. When tested with the modifications we've made since, I was able to use the full length of the tray as of late and **stack 5 blocks total**.

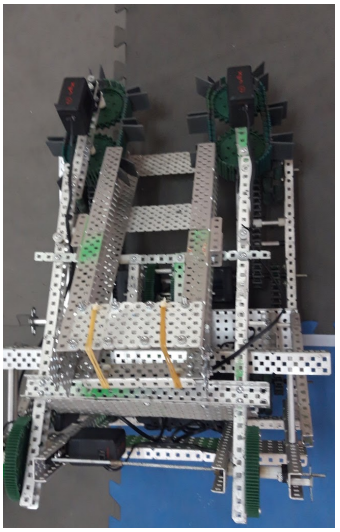
My experience with driving allowed us to **see some key points within the robot that we could continue to work on** in order to increase our stacking capabilities. Firstly, the fifth cube has trouble intaking due to the lack of tension exerted by the intake rollers. Secondly, there is limited space at the moment for the full five cubes. In order to be able to intake and stack these efficiently, more space for the tray is needed.

As a result, we spent the latter half of the day **experimenting on an extension of the tray**. This allowed it to hold more cubes such that the robot can store and stack higher. This was achieved using two 3 x 5 hole plates connected to the two parts of the tray. These had a screw and regular nut holding them together to create an axle of rotation. Rubber bands were then used to connect the two halves of the tray in order to provide tension to release the tray when it reaches a certain angle. This allowed for the second stage of the tray to be compact enough when starting the match, but extend high enough to accommodate more cubes, as seen in the next table. More experimenting and testing still needs to be done to find the most optimal design.

Retracted	Extended
	







The robot with the newly added second stage extension of the tray

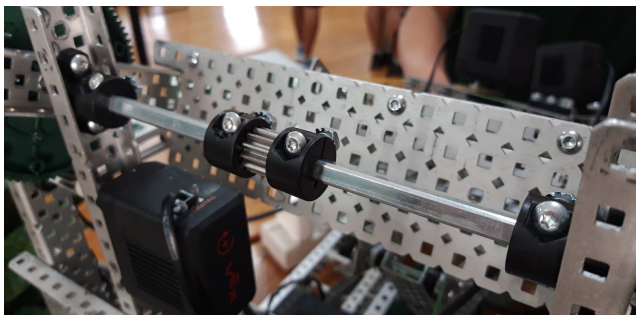
September 23, 2019 (Eion)

Summary:

1. Replaced the formerly regular-sized horizontal axle connecting the vertical support poles and the arms with a very thick axle
2. Intake mechanism folding system prototype

We first replaced the thin support-arm axle with a thick one. To do this, we drilled a hole through all four of the vertical support poles and thus inserted the axle through

them, as well as the gears to lift the arms. This is very important, as **this axle supports the entire weight of the arms with the lifting mechanisms, as well as any of the blocks it will carry**. Additionally, this thick axle fit into the drilled holes very tightly, which increased stability and reduced the chance of significant wear and tear, which may cause problems later on. The thick axle was centered exactly at the same place as the small, older axle was. Next, we worked on the intake mechanism folding system prototype. This consisted of two main parts:



- **The rotating screw lock**: the lifting/intake mechanisms and the arms were connected by only one screw with a rotating lock. As such, this would easily become loose and the connection could easily move around and away from its desired position.

- **Rubber bands and vertices**: The "vertices" were the points at which the rubber bands were tied around. These consisted of a short, wide white spacer next to the screw head, as well as a tall, thin black spacer next to that. This allowed rubber bands to be connected to fixed nodes, with which we would control the function of tension with relation to the intake mechanism's position, and thus, the movement and final position of the mechanism.

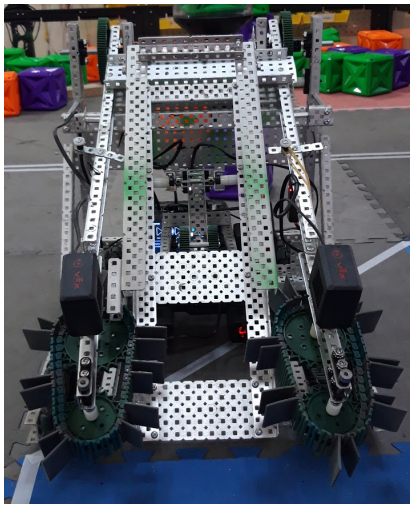
This design was unique because, instead of folding inwards from the outside, this instead did the opposite. However, the inside of the robot already had a ramp and, thus, the positioning of the arms were a bit awkward. However, the arms were able to snap into place after being pushed by the ramp. An inner-facing protrusion was connected to the arms, to which one of the vertices was placed. This increased the tension inwards and thus kept the arms and intake mechanisms aligned and straight. **However, one problem of this design was its tendency to move outwards, as well as its instability in not staying at its desired position.**

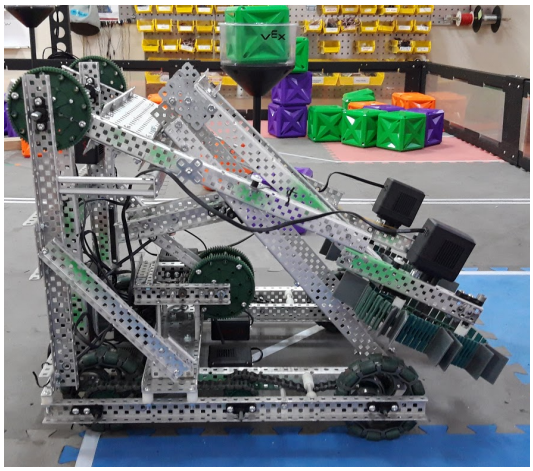
September 24, 2019 (John)

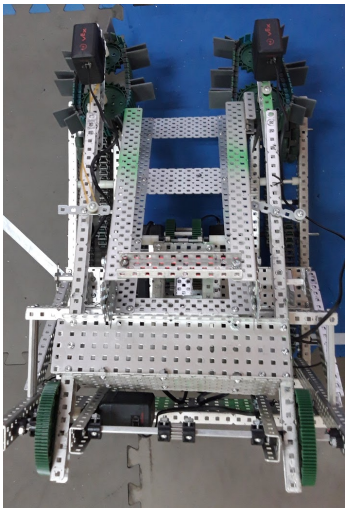
Summary:

1. Implementing new functions within the code and incorporating new hardware

Here's the status of the robot so far. The intake mechanism's design still needs some work to fit the size constraint but to test its function it has been attached in a fixed position for now. The main parts essential to its function are: The base, the arm, the intake, and the angle-shifter.







The code is **divided into various *pragma regions***: HelperTools, actions, and compTemplate.

- The HelperTools are a set of functions designed to make the autonomous programming a lot easier. It includes a function *todeg()* that converts a distance in centimeters to wheel degree rotations, a *forward()* and *turn()* command, and other implementations of said commands. These allow the program to be more concise and easier to follow while maintaining the same functionality. This also speeds up the development process by cutting down on lengthy code.
- The *actions* are a set of programs that involve the actual movement of the code. There are some work-in-progress ones such as arcade() and release() but the

main ones are tank(), which was explained earlier, and arm(). Arm() involves three parts, the intake rollers, the angle-shifter, and the lifting of the arm. The code for the first two are working properly at the moment with little possible improvements. However, the arm() code still needs some more work as the arm moves up slowly. Setting it at >0.5% is too powerful while setting it less just makes it drop. This may be due to the robot rounding it in 0.5% increments.

- The *compTemplate* is just the section involving the competition template. It consists of main() and its implementations of usercontrol() and autonomous().

Because of the transition from RobotC to V5, no external encoders need to be attached to the robot and thus we could focus more on the actual actions and the programming aspects.

The full-version of the updated code is stored in the appendix. That said, the following are the rudimentary programs for the added hardware used at this date. They are all manual and do not yet involve the use of sensors(encoders).

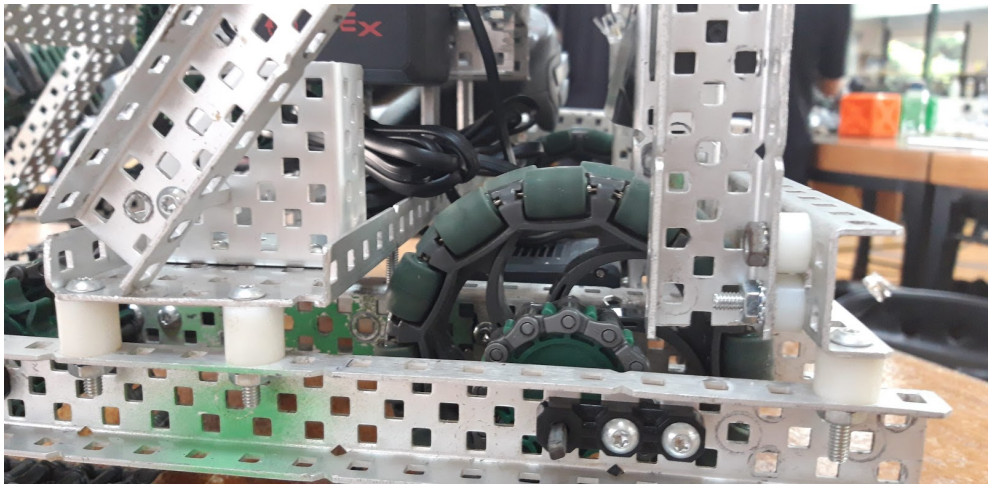
```
void tank() { //*0.75 makes the max velocity 75%
 LeftFront.spin(vex::directionType::fwd, controller1.Axis3.position()*0.75, vex::percentUnits::pct);
 LeftBack.spin(vex::directionType::fwd, controller1.Axis3.position()*0.75, vex::percentUnits::pct);
 RightFront.spin(vex::directionType::fwd, controller1.Axis2.position()*0.75, vex::percentUnits::pct);
 RightBack.spin(vex::directionType::fwd, controller1.Axis2.position()*0.75, vex::percentUnits::pct);
}
void arm2() {
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLift.setVelocity(100,vex::percentUnits::pct);
 ramp.setVelocity(30,vex::percentUnits::pct);

 //controls the ramp
 if(controller1.ButtonUp.pressing()){//from rest to perpendicular
   ramp.spin(vex::directionType::fwd);
 } else if(controller1.ButtonDown.pressing()) {//from perpendicular to rest
   ramp.spin(vex::directionType::rev);
 } else {
   ramp.stop();
 }
 //controls the rollers
 if(controller1.ButtonL1.pressing()) { // makes the rollers intake blocks;
   armLeft.spin(vex::directionType::fwd);
   armRight.spin(vex::directionType::fwd);
 } else if(controller1.ButtonL2.pressing()) { //spins the rollers in reverse / outtake
   armLeft.setVelocity(100,vex::percentUnits::pct);
```

```
    armRight.setVelocity(100,vex::percentUnits::pct);
    armLeft.spin(vex::directionType::rev);
    armRight.spin(vex::directionType::rev);
  } else {
    armLeft.stop();
    armRight.stop();
  }
  //controls the arm
  if(controller1.ButtonR1.pressing()) {//raises the arm
    armLift.spin(vex::directionType::fwd);
  } else if(controller1.ButtonR2.pressing()) {//lowers the arm
    armLift.spin(vex::directionType::rev);
  } else {
    armLift.stop();
  }
}
```

September 25, 2019 (Seba)

Summary:

1. Moving the arm to fit the robot into size restrictions

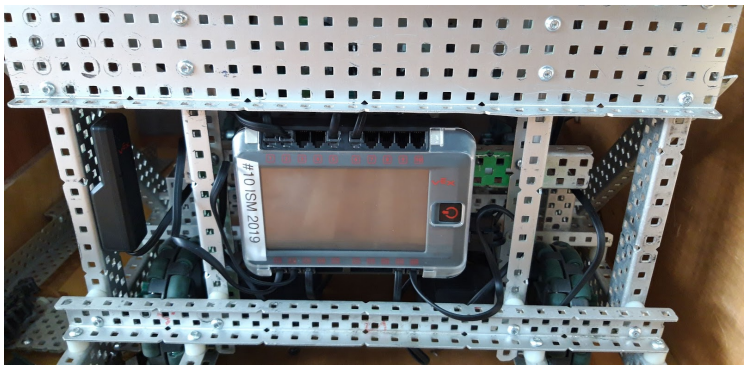


The entirety of this session primarily focused on **moving the arm forward by one space**. We had noted noted in the start of the session that the robot **did not fit within the size specification** due to the gear at the top of the robot. This became the most pressing issue for us to tackle and it spent the whole day to finish. With Thunderdome arriving soon and us consisting of only three people today (due to ISM's Great Works Concert), we focused purely on this. Additionally, we moved the front wheels back by one as well to be sure that they also fit inside the specified size.

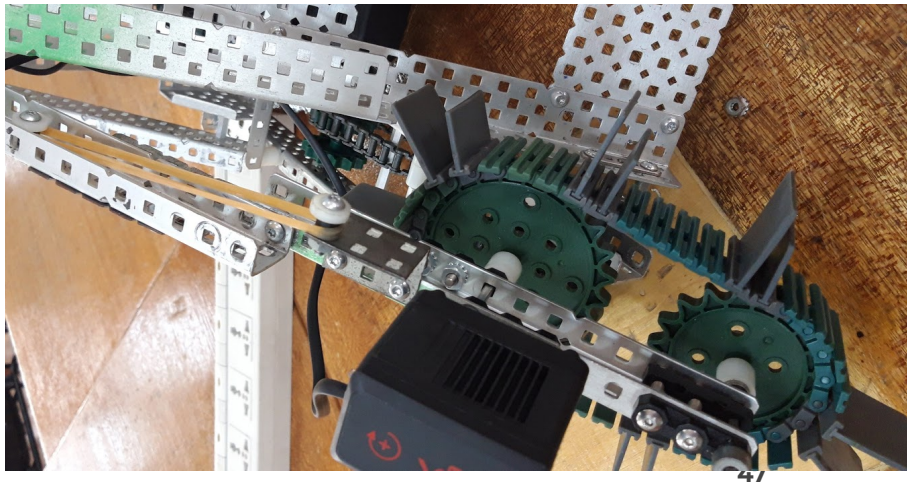
September 27, 2019 (Jason)

Summary:

1. Moving the location of the robot brain
2. Rewiring motors
3. More work on the intake retraction mechanism

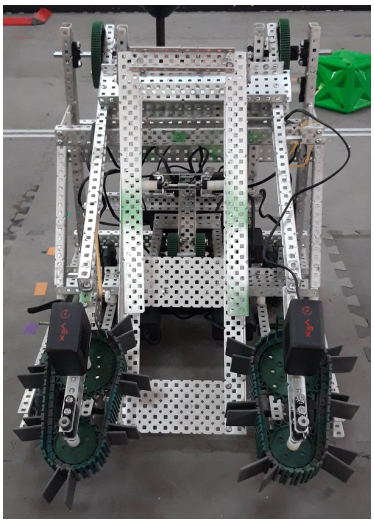
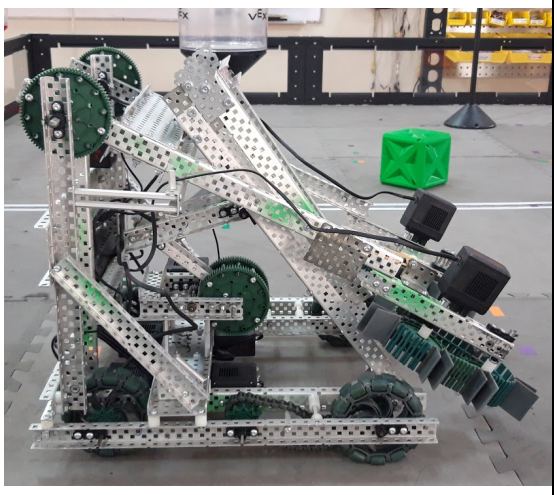
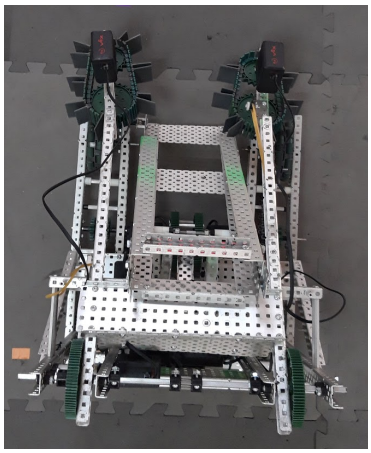


The majority of the work done today involved **renovation of aspects of the robot to make it more efficient and easier to use.** This included **relocating the position of the robot brain**, having the touch screen side facing outward. Previously the brain's LCD screen faced inside the robot so that the ports were easier to wire to the motors. This was changed by adding a c-channel and connecting the brain, with a gap between the two connected parts of the width, to face the outside. The gap created allowed for the connection of the ports with the smart cables of each motor, allowing us to more easily troubleshoot any problems. **The motors were rewired to different ports**, such that there was **as little distance as possible** to prevent entanglement of smart wires. This was also followed with the rewriting of motor ports in the main code.



Aside from this, more work was done with **experimenting with the intake rollers** for the collection of blocks. A rudimentary system involving screws was used to create a pivot point to have the motors retract. Experimenting with tension and

rubber bands was also done to aid with intaking.

The robot with the newly positioned cortex and rewired motors

September 28, 2019 (Eion)

Summary:

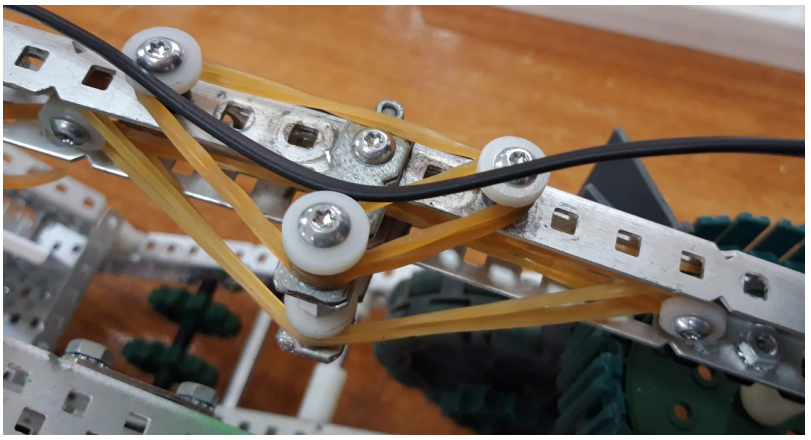
1. Fixing the intake mechanism folding system
2. Coding the angle for high post scoring

After much trial and error; after discarding many previous design prototypes, we agreed upon a newer design which seems to work pretty well for now. Some main problems we encountered before were:

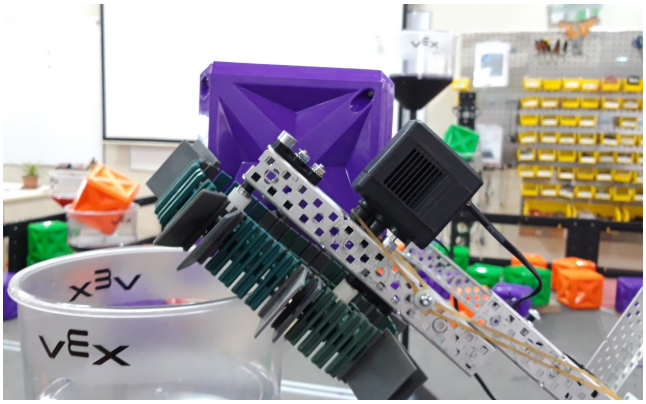
- The robot exceeding the size limitations
- The arm/intake mechanism not folding in without external input
- The arm folding in but not being able to pull in blocks because of the lack of tension keeping the arm stable, either going in or out.

The design that we decided on that looks promising consisted of a **hinge instead of using rotating screws.** This fixed the small problem of loose screws and unstable plate connections. **An array of rubber band "vertices" - the points at which the rubber**

bands are tied around - were set up across the arms, as shown in the photo. As per how this design solved some of the main problems:



- We were able to connect a rubber band to the intake mechanisms when folded outwards, thus keeping it at that state for the start of the game. This therefore kept the arms tightly folded in and within the size restrictions.
- This is still one of the main problems which we are attempting to solve. We got the arms to fold automatically, however, we still need to improve its reliability and consistency.
- With the triangular rubber band design (which will be elaborated on later), the tension keeping the arm from moving outwards was vastly increased while still allowing the arm to fold outwards without experiencing too much tension and thus, snapping. On the other hand, the hinged plate easily prevented the arms from moving too much inwards.



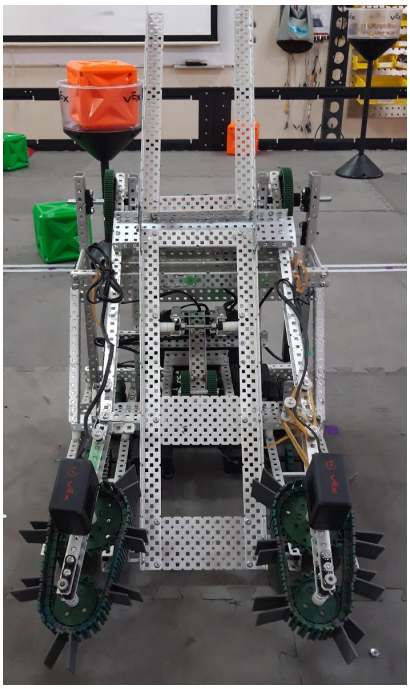
The folding of the intake mechanism is very challenging because one needs to balance the tension so that it wouldn't be too much to break the rubber bands when stretched out fully (when the arm is folded outwards), while also being tight enough to consistently and reliably push blocks up the ramp without being pushed outwards when the rubber bands are straight and not stretched out as much (when the arm it extended out to pick up blocks). This design is much more improved than any of the previous ones, as it is **both stable and automatically folding**. However, we still need to improve the folding mechanism to work even with the arm folded tightly outwards.

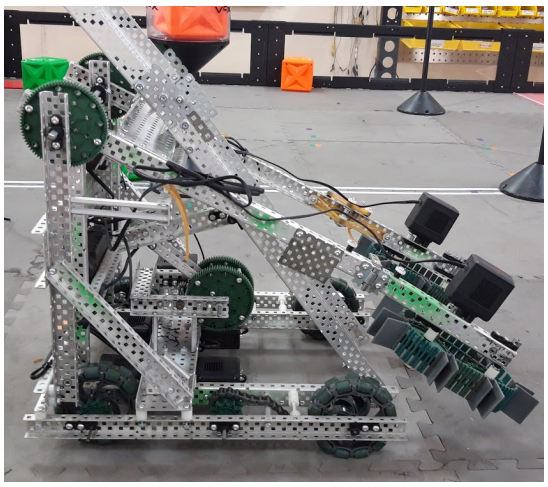
Furthermore, John **coded the robot to take the specific angles of rotation for the arm motor to automatically set the height for scoring** on high goal posts. By using the built in encoder of the motor, he was able to derive that the angles needed were 690 degrees for the alliance and small posts, and 950 degrees for the medium posts. With our current setup, we are not able to reach the highest middle post.

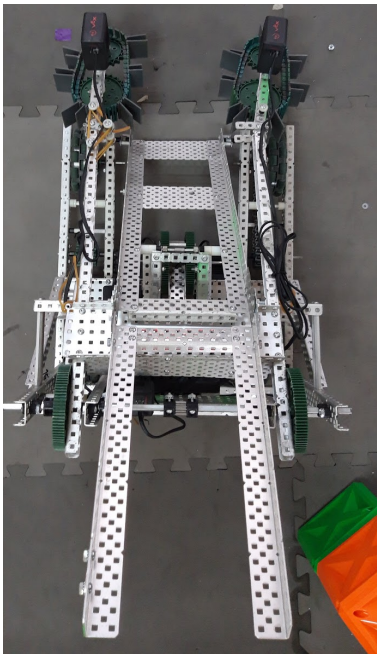
John: This also patches the previously mentioned issue with the arm slowly going up with the previous implementation. By using the built-in encoders to move the arm to a specified angle we don't have to guess what power is needed to maintain the position. This gives the driver easier control over the robot as they can simply choose between three arm states, each controlled by the buttons X, A, and B.

```
//armAngle is instantiated in a global scope and redefined at a local scope
double armAngle = 0;

//Changes the value of armAngle
if(controller1.ButtonX.pressing()) {
  armAngle = 950;
} else if(controller1.ButtonA.pressing()) {
  armAngle = 690;
} else if(controller1.ButtonB.pressing()) {
  armAngle = 0;
}
Brain.Screen.printAt(100,150,"Angle: %0.2lf",armAngle); //prints the value of armAngle(for troubleshooting)
armLift.setVelocity(30,vex::percentUnits::pct); //sets the velocity to 30%
armLift.rotateTo(armAngle,vex::rotationUnits::deg,false); //moves the arm
```

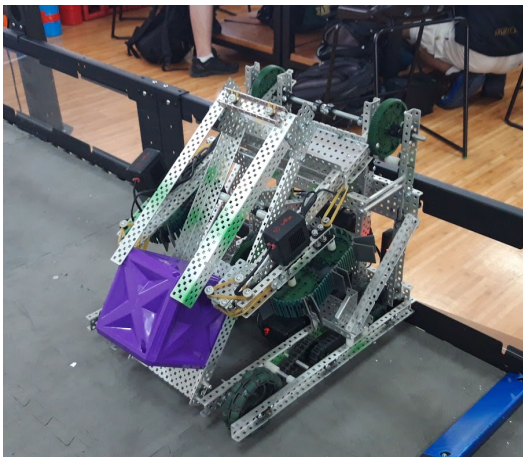
*Note: this was before we were aware of the command motorName.setBrake() for manual control applications.







September 30, 2019 (Eion)

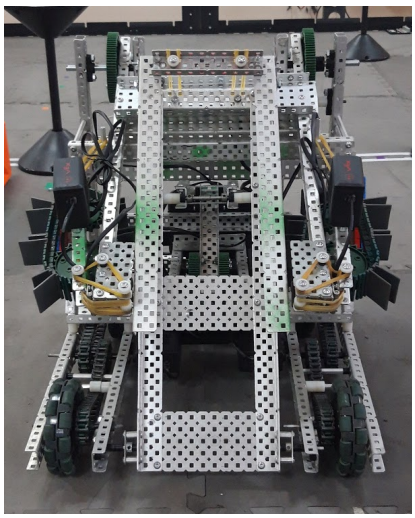


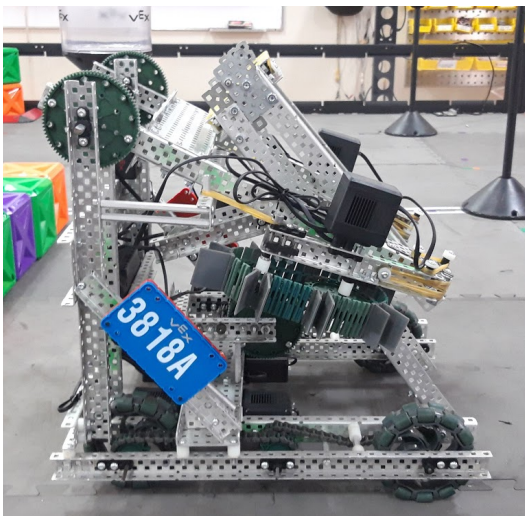
Summary:

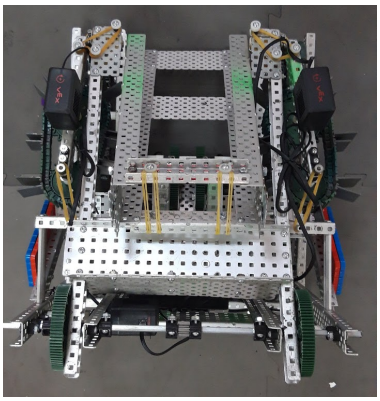
1. Toolbox
2. Ramp Alignment

We started **organizing our toolbox** a few days away from the Thunderdome competition. We included many important items such as screws, nuts, screwdrivers, tracks, wrenches, and many more. This was intended as a failsafe in case something went wrong with the robot on the competition day. However, the main issue we dealt with today was the ramp alignment. Due to the misalignment of the ramp, the intake of the blocks became much more inefficient than what we were hoping for. Additionally, this placed a lot more strain on one side of the robot over the other. This was quite difficult to fix, as there already was an inherent misalignment caused by the slight bending of the aluminum c-channels. However, by

using various spacers and washers, we were able to **lock the ramp relatively parallel to the robot's base** in the hopes of optimizing its intake. This would definitely be crucial in the driver skills and matches, but even more so for the autonomous skills. Since no micro-adjustments could be made during that round, the robot's precision must be as good as it can be. We are hoping that this adjustment will make the block intake much easier and more importantly, more consistent.







The fully retracted robot, fitting within the 18" x 18" x 18" limit

October 2, 2019 (Jason)

Summary:

1. Developed Strategies for Thunderdome
2. Practiced Driver skills
3. Practiced Auton skills

The time spent this week was spent towards **developing strategies for Thunderdome.** **Eion and Justin** were particularly responsible with **developing tactics for scoring** during matches and skills runs. **John** worked on **coding a 5 point autonomous** by intaking a preload and 4 cubes lined up towards the end of the field. **Seba** worked on **quality control and documentation**, while **I** practiced **driver skills**.

These were the Strategies for Thunderdome that we came up with:

Matches:

1. Auton: Stack 4 flat cubes and a preload (5 pts). Driver: Stack any colored cubes, then determine which color maximizes points for placing in a goal post
2. Stack a specific color of cube, then place those color cubes in a goal post

Driver Skills:

1. Stay on one side of the arena. Stack 4 cubes and a preload (5 pts), Stack another 5 cubes (10 pts), then score on 2 goal posts **[30 pts]**
2. Stay on one side of the arena. Stack 5 cubes and a preload (6 pts), stack another 6 cubes (12 pts), then score on 1 goal post **[24 pts]**
3. Stay on one side of the arena. Stack 4 cubes and a preload (5 pts), stack another 6 cubes (11 pts), then score on 2 goal posts **[33 pts]**

During **driver skills** practices, I was able to **successfully and consistently score 30 points** by doing the first driver skills strategy.

Auton Skills:

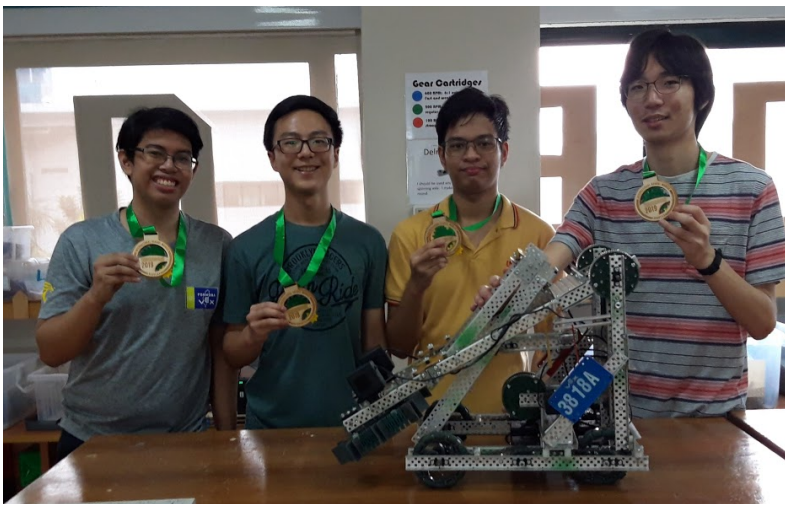
1. Stack 4 flat cubes and a preload (5 pts)
2. Stack 4 flat cubes and a preload (5 pts), then place a cube in a goal post [10 pts]

John was able to **code a fairly consistent 10 pt auton for skills** (as seen to the right, credits to Mr. Dingrando), and implement the 5 pt aspect during a match.

October 4–5, 2019: Thunderdome Prelims (Jason)

Summary:

1. 1st Place overall
    a. 1st in Skills
    b. 1st in Qualification Matches
2. Analysis of performance during matches
3. Post tournament reflections, observations, and improvements
4. Team Analysis



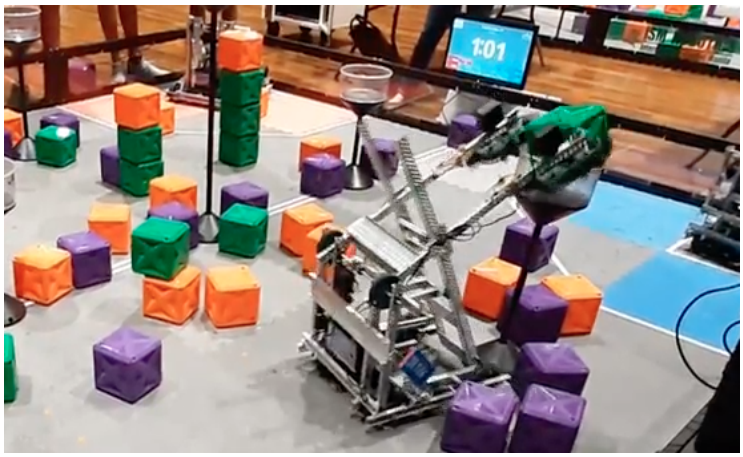
Thunderdome is a preliminary ISM competition where 8 teams (previously 9) advance to the RoboRumble tournament. Setup as a more official VRC tournament than Trial of the Pushbots, **it gave us the opportunity to do matches with teams,** given the month of work and development, **and skills challenges, both driver and autonomous.** Thunderdome allowed us to compete in a more in depth and competitive environment, and allowed us to see what we were currently good at and what improvements could potentially be made in a competition setting.

Overall, we placed First. We were First in Skills challenges, with 0 points in programming and 30 points in driver skills, scoring 30 points total. We were only given 3 trials each for both aspects. We could theoretically score 10 points total in programming, however, the autonomous did not work for all three trials, as either too few cubes were intaken (messing up the routine), misalignment which caused it to stack outside of the goal zone, and inputting the wrong code. Our driver skills points fully boosted our ranking, which came from stacking 10 cubes (10 pts) and scoring 2 cubes on goal posts (30 pts). This was the highest scoring driver skills of Thunderdome, and was greater than all other scores combined. This could be improved further by attempting to score another cube in a goal post, or potentially stacking more cubes.

We placed 1st in matches, winning 9 out of 10 rounds. The nature of matches made each one varied based on who we were with and against. We analysed and video

recorded each match to note what went well, what went wrong, improvements, and circumstantial factors. There were many times when the robot was able to stack during auton, gaining us the auton point. We were able to stack in all matches, except the first match where our ramp got stuck and did not flip back, limiting our ability to score.

The robot (bottom left) scoring 5 pts in autonomous



The robot depositing a cube on a tower

Design Journals were also graded per team out of 27, based on the Design Notebook Rubric. **We placed 1st**, with a total score of 21. Feedback gained would be used in order to maintain the same standards for the journal working towards RoboRumble, while making improvements to it as well. **In RoboRumble, Design Notebooks are a determining factor in determining teams competing at Taipei.**

Post tournament reflection points and improvements:

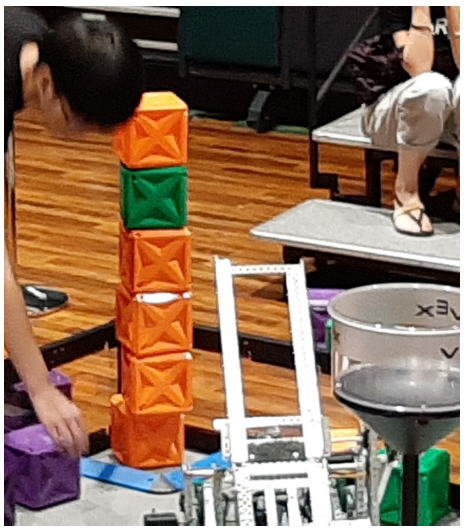
1. Use the same medium sprocket gears for both back and front gears of the intake to allow for diagonal blocks to intake easily, and to prevent blocks from being jammed in the ramp
2. Adjust the design of both the intake and the ramp to allow for stacking more than 5 cubes at a time. Potentially replace the middle portion with polycarbonate to address this. 8 cubes in a stack would be a good goal.
3. Integrate a design with the ramp to allow it to expand more and stack over 5 cubes at a time

4. Practice driving and scoring more to be able to stack 2 towers on the large goal zone
5. Experimenting with auton modifications:
    a. A vision sensor to make the auton towards the goal zone more efficient and accurate
    b. Employing a new strategy to make the robot shift right and left (while moving forward) to ensure the stack centers in the goal zone
6. Finalize the retracting design and mechanism for the intake rollers to prevent them from interfering with stacking

Strats/Considerations:

1. Cut down seconds during driver skills to be able to score an extra cube in a goal post, or potentially create another stack.
2. Place more emphasis on colors of cubes to stack during a match. Coordinate more efficiently with the match coach.
3. Utilize the Purple Cubes effectively after being given them with the auton bonus.

TEAM ANALYSIS #2



Only 8 teams advance from Thunderdome and none were eliminated, apart from the disbanding of Johnny Boi. Thunderdome gives us the opportunity to see how well teams that passed did, and **we were able to see their strengths and weaknesses**, and attempt to **determine what focus they would potentially aim for** leading up to RoboRumble. As seen previously, most teams aimed at using the tray design to stack and score towers of cubes, while only 4 teams (Mad Max, EDSA, Munchlax, and Athena) could score on goal posts. Munchlax was particularly surprising for being the only team to stack 6 points. As such, we did further analysis on teams during the competition.

Rank	Team	Skills Rank	Match Rank	Strengths	Weaknesses	Future functions/potential
1	3818 A Mad Max	1	1	• Consistent stacking during driver and auton (5 cubes high) • Can score cubes on goal posts	• Auton can be inconsistent • Some mechanisms get stuck (ramp), which can impact performance	• Improvements and practice with robot • Tweaking of auton • Higher capacity for stacks of cubes
2	3818 F Munchlax	3	2	• Can stack 6 cubes high • Functioning stacking auton • Reliable alliance partner • Can place cubes on low goal posts	• Auton is inconsistent • Placing the towers in the goal zones is very inconsistent • Very easily knocks over towers while withdrawing • Inexperienced driver	• Improvements and practice may make scoring more efficient
3	3818 J C5 Extension	2	6	• Consistent 1 pt auton	• Only has pushbot capabilities	• Potential addition of tray and roller intakes
4	3818 H NAIA na Lang	6	4	——	• Only has pushbot capabilities	• Potential addition of tray and roller intakes
5	3818 G Euphoria	8	3	——	• very inexperienced driver • Has difficulties both towering and stacking in goal zones	• Driver needs considerable practice • Many changes to the intake/ramp needed
6	3818 B EDSA	4	7	• Can stack 5 cubes high • Experienced driver (Worlds Team) • Can place cubes on low goal posts	• Inconsistent scoring of towers in goal zones	• Improvements and practice may make scoring more efficient

7	3818 C Cream	7	5	——	• Intake mechanism does not function • Cannot stack more than 1 cube high	• Will need significant revisions to current intake/ramp system • future potential is somewhat uncertain
8	3818 E Athena	5	8	• Can stack 6 cubes high • Can score and descore cubes on goal posts	• Very inconsistent mechanisms and driving • lack of proper ramp system means that cubes spill over	• Working on a wall to block scoring zones • May become more consistent in the future

NB1. Although all ISM teams advance, **10 other teams will also be taking part of RoboRumble**. 5 teams from Taipei American School, 1 local team from the Lyceum of the Philippines University HS, and 4 teams from Harrow International School Hong Kong. Although team analysis for ISM teams was made, it is important to take into account the uncertainty that these other competing teams bring to the competition (and how knowledge on their capabilities could be used on the day of RoboRumble).

Below is the full list of the results from Thunderdome. There was a considerable gap between our team and others, however, we expect teams to improve leading up to RoboRumble.

Thunderdome Results												
No.	**Team**	**Matches**				**Skills**				**Overall**		W/L Ratio
		WP	AP	SP	**Match Rank**	P HS	D HS	Total	**Skills Rank**	**Score**	**Rank**	
1	3818 A Mad Max	18	45	59	1	0	30	30	1	2	**1**	90%
6	3818 F Munchlax	12	18	74	3	0	6	6	2	5	**2**	60%
8	3818 J C5 Extension	16	39	44	2	1	3	4	6	8	**3**	80%

7	3818 H NAIA na Lang	4	18	64	6	1	0	1	4	10	**4**	20%
9	3818 G Euphoria	2	3	40	8	0	6	6	3	11	**5**	10%
2	3818 B EDSA	12	30	65	4	0	4	4	7	11	**6**	60%
3	3818 C Cream	4	9	50	7	0	1	1	5	12	**7**	20%
5	3818 E Athena	8	24	84	5	0	2	2	8	13	**8**	40%

Since Design Notebooks are a determining factor for ISM Vex Formosa Teams, we have outlined the potential placing of each team with Notebooks taken into account. This slightly changes the dynamic of placings of the teams, as seen below.

No.	Team	Match Rank	Skills Rank	Notebook Rank	Total Score	Rank
1	3818 A Mad Max	1	1	1	3	**1**
6	3818 F Munchlax	3	2	6	11	**2**
9	3818 G Euphoria	8	3	3	14	**3**
5	3818 E Athena	5	8	2	15	**4**
3	3818 C Cream	7	5	4	16	**5**
2	3818 B EDSA	4	7	5	16	**6**
8	3818 J C5 Extension	2	6	8	16	**7**
7	3818 H NAIA na Lang	6	4	7	17	**8**

October 7, 2019 (Eion)

Summary:

1. Cleanup
2. General Repairs

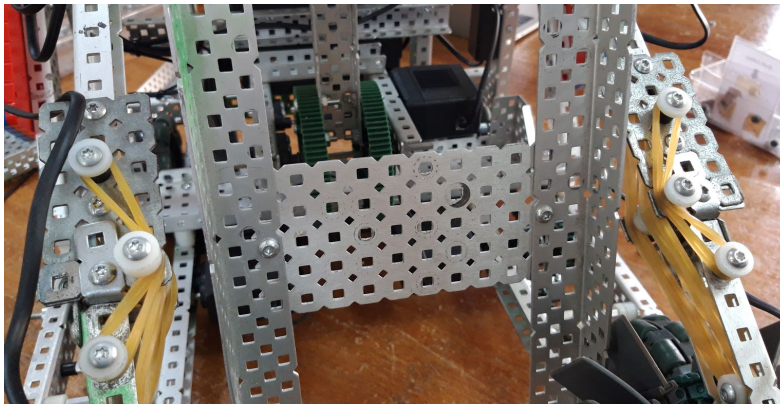
October 7 was the assigned **cleanup day** right after the Thunderdome. As such, the **materials from the toolbox were returned.** However, we took note of these in order to ensure that they were brought to Roborumble as well. Preparations from now on would be focused on **optimizing the stacking capabilities and intake consistency** of the robot. The first in doing so was to conduct the general repair of the robot. This was thoroughly observed and fixed, part by part. We noticed that some of our screws and nuts fell off during the Thunderdome matches and wanted to prevent this from happening in future rounds to reduce the chance of malfunction. However, this only served as a safeguard since we already placed backup systems within the robot in case of this type of failure. Some of these backup systems were redundant screws, reinforced c-channels, and lubricant for the motors.

October 10, 2019 (Eion)

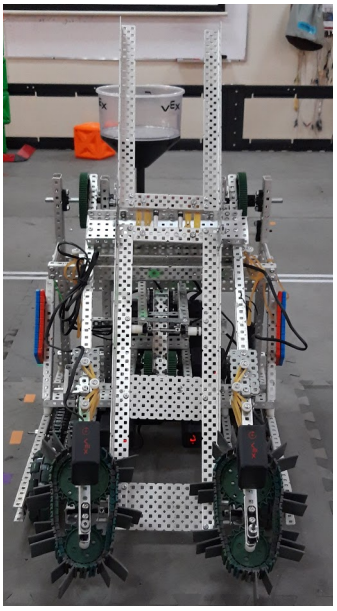
Summary:

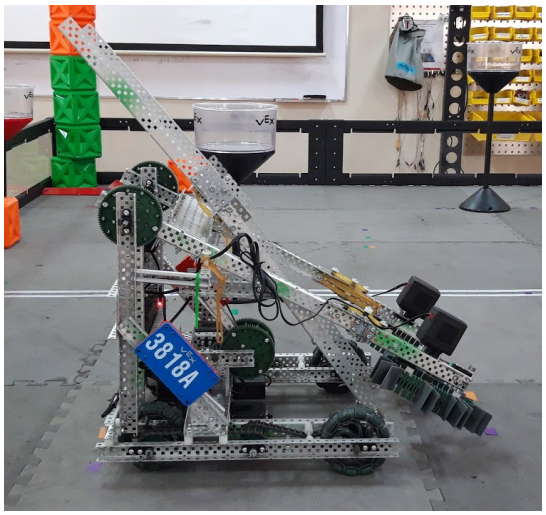
1. Optimize tray design
2. Optimize intake flipping mechanism

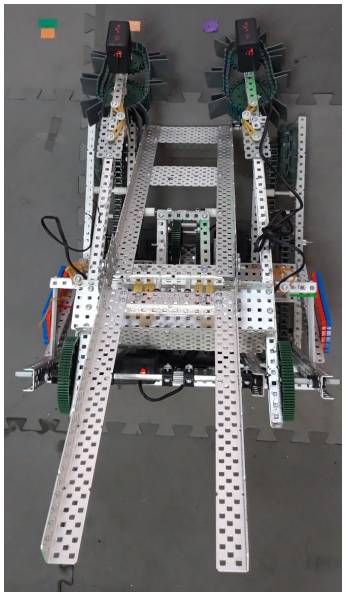
A big part in optimizing the tray design was **adding "ribbing" or plates between the sides of the ramp**. This ensured that the blocks didn't end up stacking diagonally and thus hindering the progress or even preventing the intake of other blocks up the ramp. This was a somewhat simple process. Furthermore, the flipping of the second section of the



ramp was tested repeatedly to ensure consistency. The design of the hex nut hinges that allowed the ramp to flip out its second section was also tested. In the end, the original design was kept. In addition, the intake flipping mechanism prototype was further optimized. This was done by moving the center vertice up the arm of the robot, thus maintaining the tension keeping the arms in towards the center while extended and simultaneously keeping the intake mechanism free to swing in when released by reducing the tension while the arms were folded outwards. When tested, this worked very well. The arms were ensured to stay folded at the starting position using our previous design of the rubber band connected to the treads. When the intake mechanism motor moves, the rubber band slips off, thus releasing the intake mechanism to fold inwards.







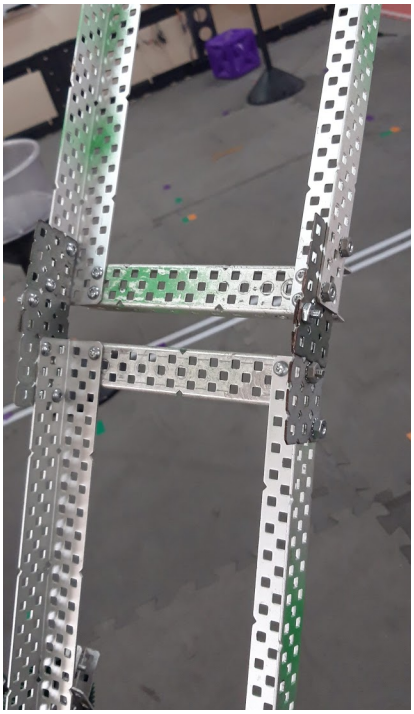
Front, side, and top views respectively of the robot with both the ramp and arms extended

October 12, 2019 (Eion)

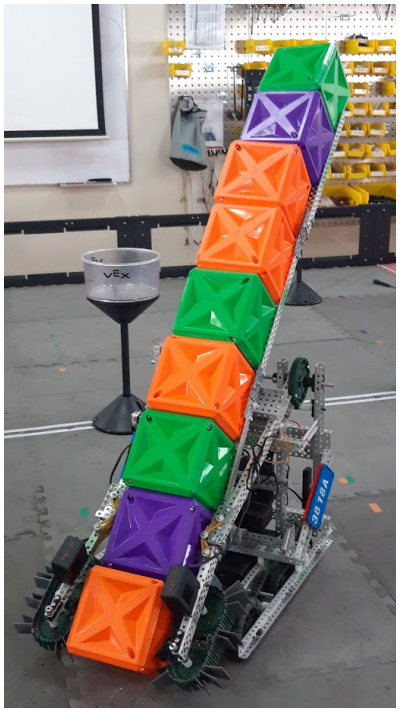
Summary:

1.) Experimenting with section 3 of ramp (basic flipping mechanism)
2.) Replacing stripped wheels

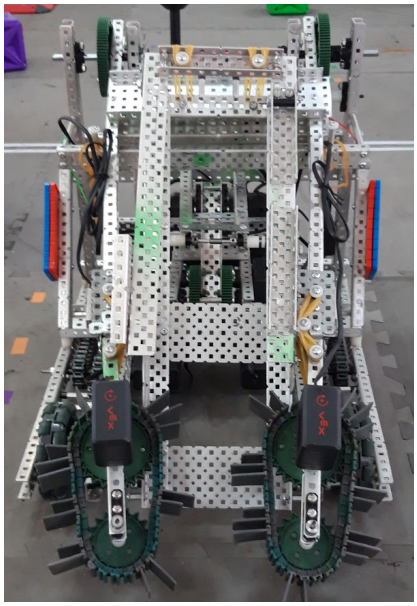
Today we **experimented with stage 3 of the ramp**, creating our first physical prototype. We designed this prototype to be a basic flipping mechanism, similar to that of the one connecting the first and second sections of the ramp. One problem with this, however, was that unlike the main flipping mechanism, this could not be propped up to prevent the robot from exceeding the size limit. Thus, our design goal was to keep the third layer's protrusion outwards when folded at a minimum to keep it within the size limitations. This proved to be a challenge, as our robot was already quite close to the maximum width and length of 18 inches. In the end, we decided to test our other designs, as many other prototypes also seemed viable. Additionally, we noticed one of our **wheels became stripped and subsequently replaced it** to avoid failure on the field. By using more reliable parts, we hope to ensure a greater degree of consistency to the robot's performance, thereby enabling us to more effectively stick to the optimal match strategies we prepared before the match.

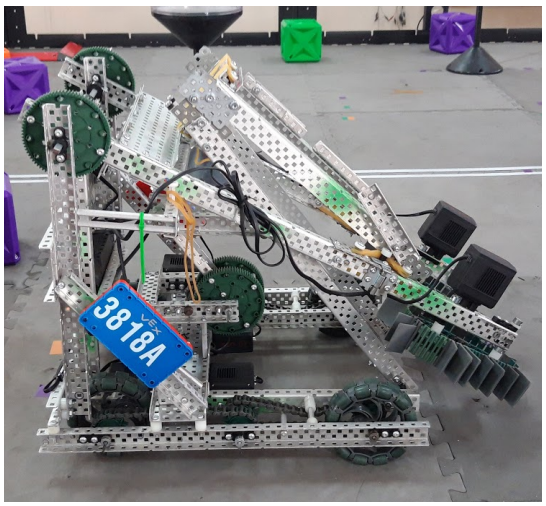


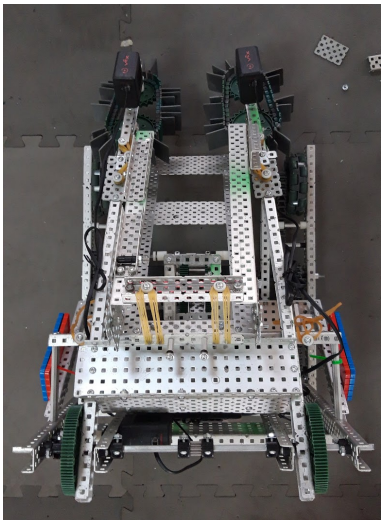
The additional 3rd stage test, using the same flipping mechanism as before



The robot carrying 9 cubes with the addition of the 3rd stage tray







Progress done on the robot with the 3rd stage detached

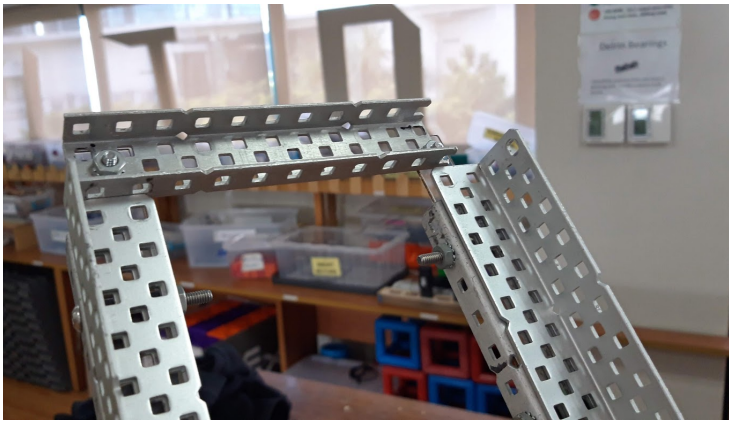
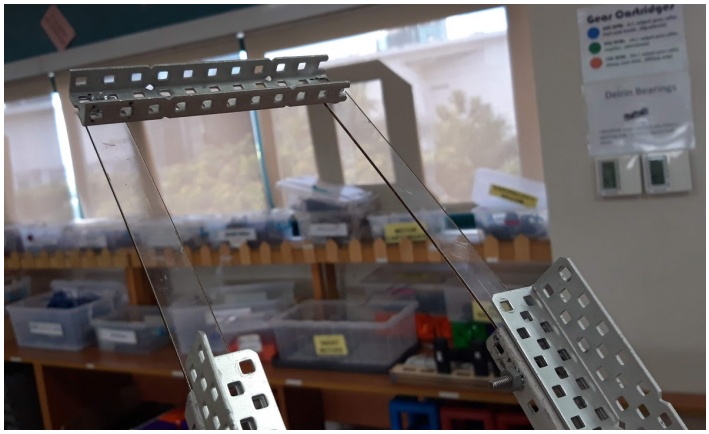
October 15, 2019 (Eion)

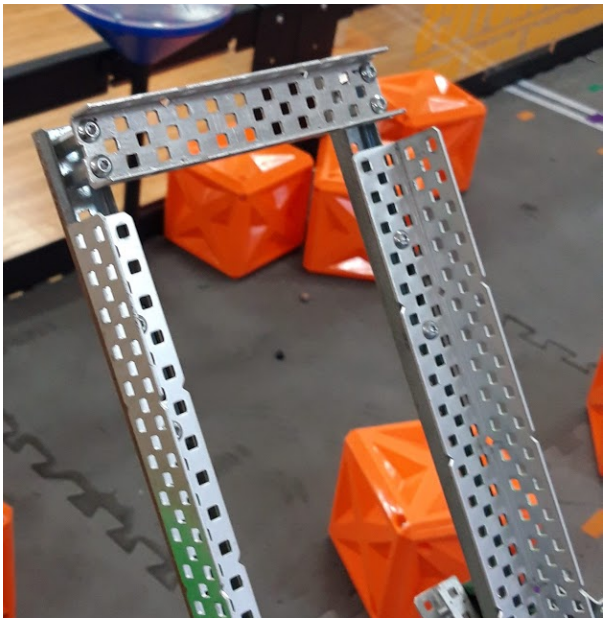
Summary:

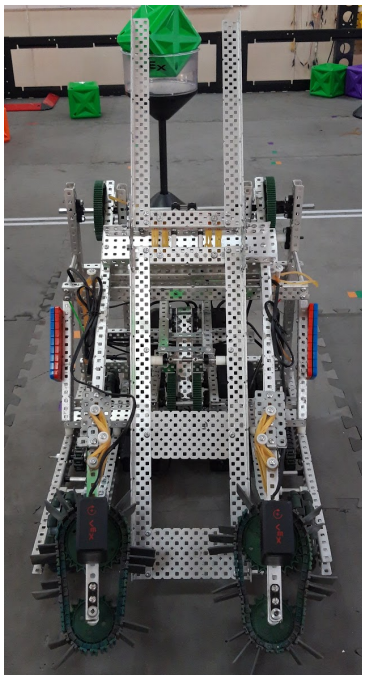
1. More experimentation with section 3 of ramp (linear sliders and polycarbonate)

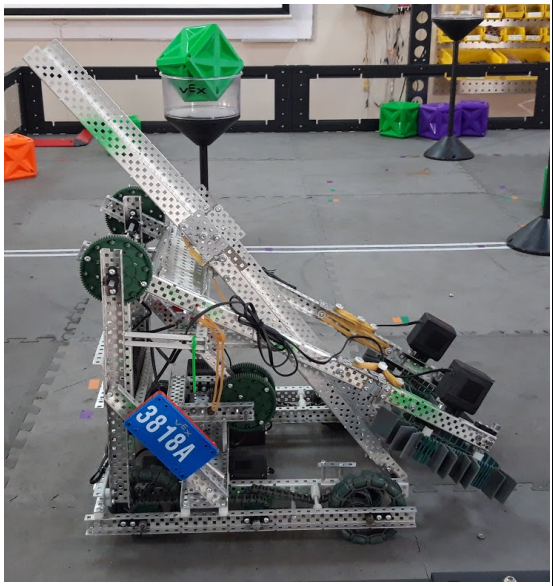
Our main focus was to create a **working third stage of the ramp** in order to be able to stack towers 8 blocks tall, instead of just 5. This difference of 3 blocks really places a huge advantage in our favor, as it creates a 60% increase in the number of blocks that can fit within the goals. This difference of 3 blocks would therefore build up as more blocks are scored within goals and on towers. For example, if 3 stacks are scored and 2 blocks are placed on towers, this will give 27 more points for skills. As such, we now attempted to create a prototype consisting of linear sliders and polycarbonate. Our intention was for the third layer of the ramp to slide outwards to create more space for the blocks. However, the main problem we ran into was the stability, or lack thereof, of the polycarbonate surface. It could not reliably carry the weight of the blocks. In addition, if the sliders would fall back onto the start position, we would have a big

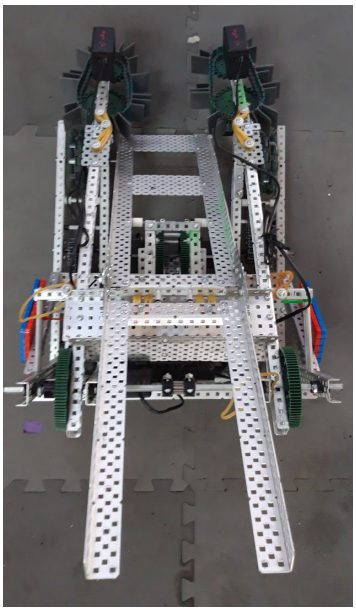
problem as it would severely damage our stacking capabilities for the match. Thus, we decided to try out some of our other prototypes before deciding on which one to use.

Polycarbonate Trial	
Retracted	*Extended*
	

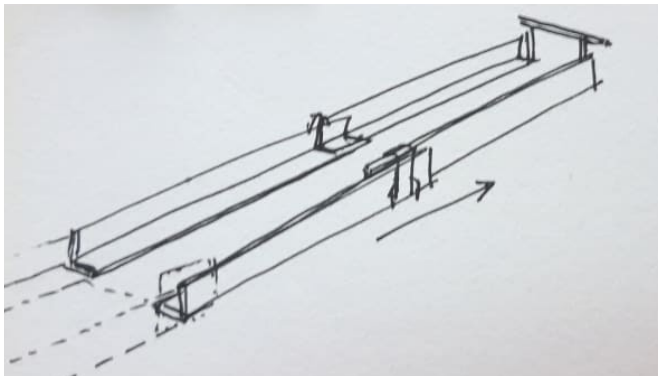
Linear Slider Trial	
Retracted	*Extended*
	







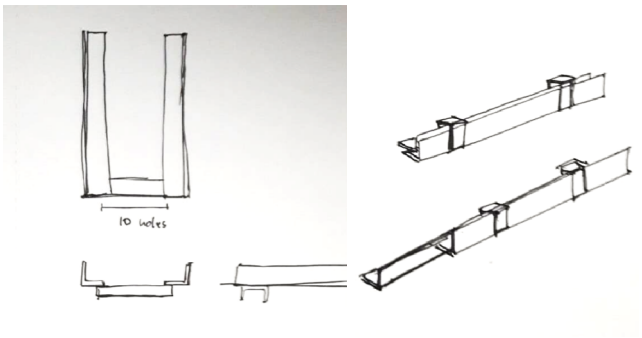
October 17, 2019 (Jason)

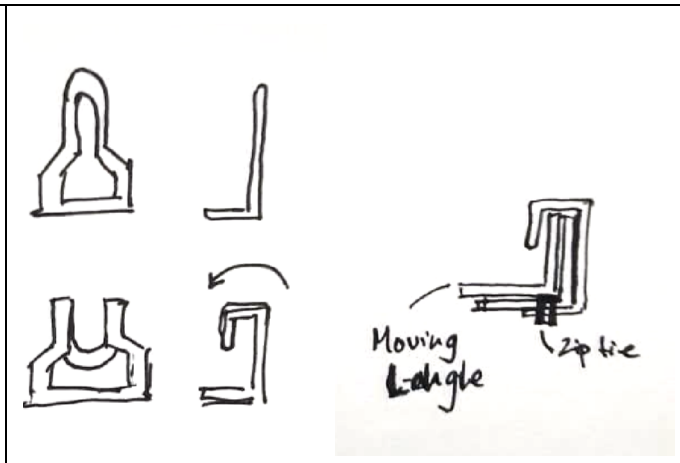


Summary:

1. Further experimentation with section 3 of ramp (angle gussets)

The time spent today was focused on researching and developing different viable designs to use for the third stage of the tray. The main idea that we came up with was the use of angled gussets to create a sliding mechanism for the tray such that it would fit when retracted. This required us to bend these angled gussets in order to hold the additional angle plate to be used to extend the mechanism. The rest of the session was spent building the mechanism itself. We determined that there was too much friction between the gusset and the angle plate such that it required a great amount of force to push up. We went back to experimenting with the 3rd stage flip after this attempt.

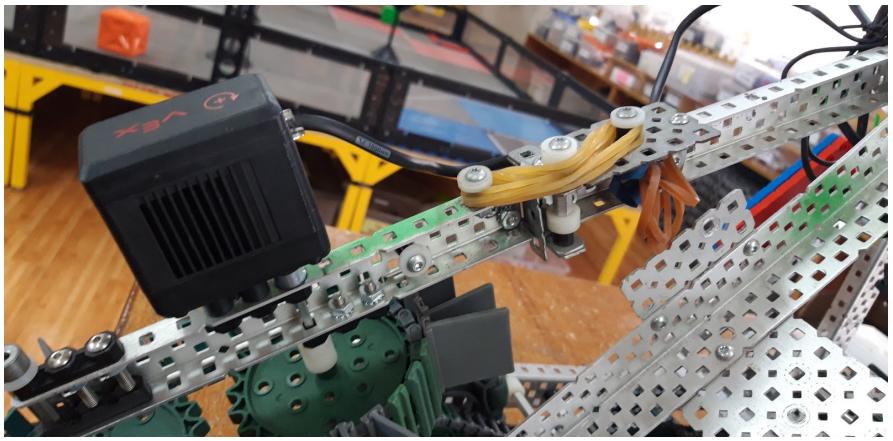




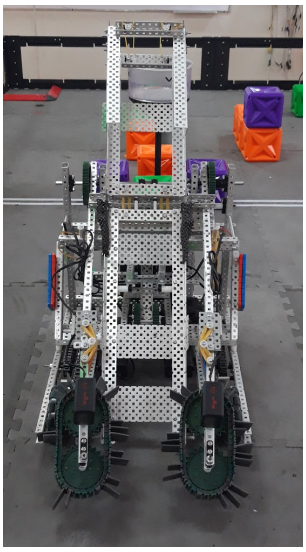
October 24, 2019 (Justin)

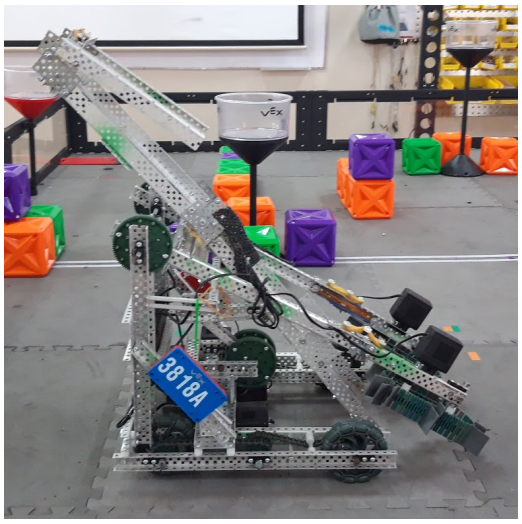
Summary:

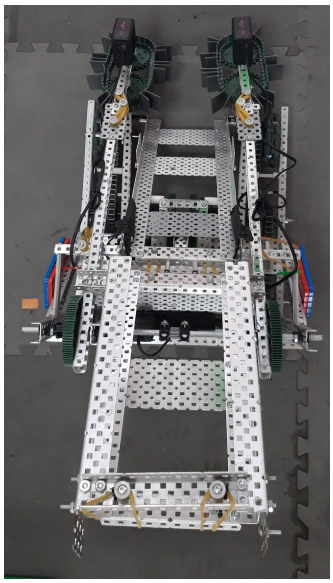
1. Moving the arm intake mechanisms higher



Today we were primarily occupied with **moving the intake motors up** by several holes in order to **provide greater surface area and stability** for cubes on the ramp surface. This was a very time-consuming endeavour which took up most of the build session. After making the changes, we performed some trial runs in the arena and found that, as expected, cube intaking performance was enhanced.







October 25, 2019 (Justin)

Summary:

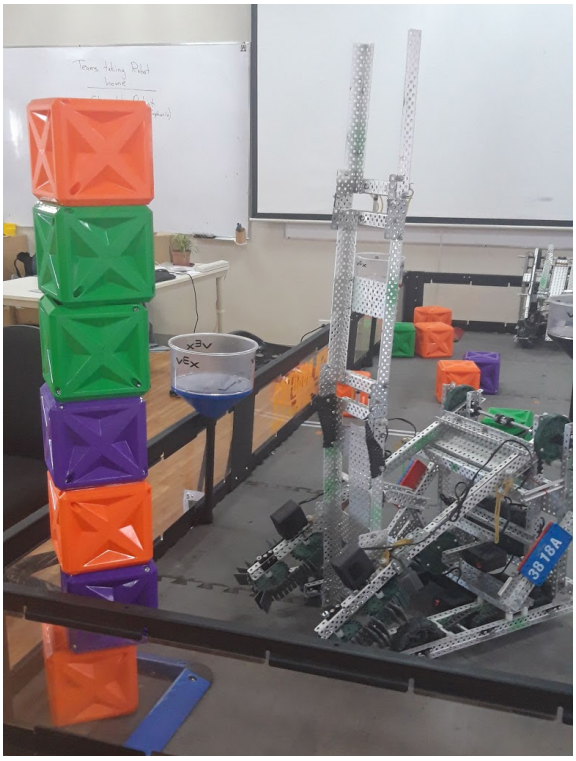
1.) General maintenance
2.) Driver skills practice
3.) 7-point autonomous

For today, we didn't want to attempt anything of too ambitious calibre because it was almost the school break and we wanted to save any major changes for later on. As a general rule, we limited ourselves to very minor repairs/maintenance and practicing robot drivers/auton – we went back and examined the entire robot to make sure that all loose screws and axles were tightened.

During the driver skills practice, we faced issues with deploying the third, topmost stage of the ramp. There was a very tight compromise between having the ramp docked low enough so that it just fit in size restrictions and simultaneously having it raised high enough at a certain angle that intaking cubes would push up and help the ramp deploy. The issue was that if the ramp was docked too low, intaking cubes would simply push ineffectually against the ramp (which presented only a flat surface to push against) without making it deploy. One of our most important tasks in the coming days was to create a system for reliably deploying the third stage.

Given these issues, we mostly practiced driver skills in a state where the ramp was already deployed. We attempted to intake 7 blocks, but approaching the final 1 or 2 blocks, the intake rollers seemed to be pushed outwards without applying sufficient pressure on the cubes. Only by directly pushing against the walls of the arena, could we intake the final few cubes. It seemed that there wasn't enough rubber band tension nor motor strength. During the break, we hoped to fix these issues by switching to red-labelled motors (which have 3 times higher mechanical advantage than existing motors, trading-off speed) and replace rubber bands. Thinking in the long term, we hope to install a more reliable system, perhaps an automatic locking system involving screws/axles, for fixing the intake motors in place without using rubber bands. Maintaining sufficient compression was key for intake reliability.

Finally, we developed a new autonomous code to complement our robot's greater scoring capabilities. The robot would be positioned further back to intake the 2nd last line of cubes; after intaking, it would rotate right, and drive backwards at a 45° to position itself to intake the other 4 line cubes of closest to arena wall. Due to the same intake issues we faced above, the robot was unable to score the intended 7 cube pieces consistently.



The robot, shown scoring 7 points during autonomous

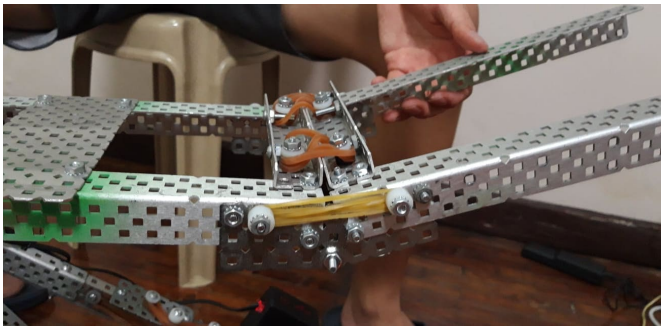
October 29, 2019 (Eion)

Summary:

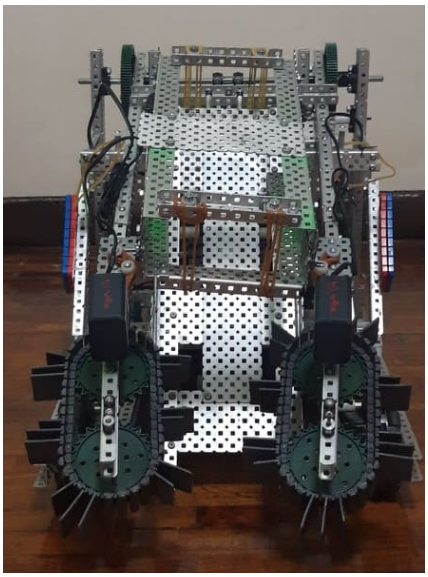
1. Replaced the motor torques
2. Creating a working prototype for section 3 of the ramp
3. Optimizing intake tread mechanism
4. Automatic ramp and intake mechanism movement
5. Journal review

On October 29, Jason, John, and I met outside of school to work on the robot design and the design notebook. First, we replaced the intake mechanism motors, changing the green motor cartridges to the red ones in order to increase their torques. We found that this worked really well in pulling the blocks in, as previously, the green motors had trouble intaking the 7th and 8th blocks due to the lack of torque. Now, the robot could consistently intake them to fill up the third layer of the ramp. However, this did decrease the intake speed, which consequently required the robot to get closer to the blocks to avoid pushing them towards different directions due to the slower movement of the treads. In the end, we decided that the new design of using red motors would work better, as the extra two blocks we could fit in a stack would have a large impact on our scoring capabilities. We then moved on to the ramp's third section's design. We had another failed prototype, in which we tried to replicate the rubber band vertice strategy that worked so effectively with the intake mechanism on the arms. However, in the end, we realized that this would not be physically possible with a simple 3-vertex design since there was no extra space or plates protruding from the hinges in order to keep the robot within bounds. This meant that there would always be less tension when the ramp was closed as compared to it being slightly open. The "breakthrough" in the ramp design came later on in the form of a simple realization. We left the third

section of the ramp slightly propped up, using the rubber band design similar to that of the one connecting the first and second sections of the ramp. This was made possible by some slightly protruding screws on the ramp face. As such, as blocks were taken in, there was now enough momentum to flip the third layer of the ramp out when the second layer flipped out.



The reason why this worked while the other previous flipping prototype didn't was simply the minimization of the protrusion; removing all the small unnecessary factors adding to the length of the third layer when folded kept us within bounds while being able to prop the third layer of the ramp upwards as well. The arm intake flipping mechanism already worked well after choosing the best option out of our many tested prototypes, so now we focused on the treads. We made sure that the treads had the exact same pattern and frequency on both the right and left intake mechanisms in order to reduce the difference between the pulling forces on the two opposite sides. This would hopefully ensure that the block would go straight up and thus not get stuck diagonally on the side plates connected to the ramp. At the end of all the optimizing, both the arm intake mechanism and the ramp could flip out reliably all while keeping the robot within bounds at the beginning. Lastly, we worked collaboratively on the journal, discussing which sections we were missing and which sections needed to be improved.



November 4, 2019 (Justin)

Summary:

1. Driver skills strategy and practice

During today's shared study hall session, Jason and I practiced driver skills. Compared to before, the robot was much more successful in intaking cubes. I believe that the stronger compression/grip provided by the stronger new red-label motors was an excellent tradeoff. However, the robot faced intake issues when the cube stack exceeded about 5 blocks – getting stuck in the middle of the ramp until a sufficient pressure broke it free. We hypothesized that the friction mats and/or screws installed on the ramp was causing the cubes to get stuck.

During the afterschool build session, Seba and I were tasked with creating a mechanism for deploying the ramp's third stage. We tried/considered a variety of possible solutions throughout the entire build including:

1. Connecting the arm motor axle to the top portion of the ramp (and other pulley systems)

    a. This failed because the rubber bands did not have enough leverage at the rubber band's point of contact – akin to how it is difficult to open a door by pushing near its hinges
2. Attaching 90° bent bars to the arms to lift up the ramp
    a. The arm motors did not have sufficient speed to raise the ramps
3. Installing a torsion-powered puncher/crossbow to 'punch' the ramp into upwards deploying-motion
    a. This was abandoned because partly due to space constraints on the ramp (do not want to block intaking cubes) and dangers of inconsistent and overengineered solutions

The difficulty of developing a deploying mechanism came from the fact that the ramp required a very short and powerful jabbing force in order to deploy – unlike the very gradual and slow forces provided by pulley systems/arm-assisted raises. Hence, at the end of the session, I (Justin) began seriously considering to revert back to a linear slider design.

A trial skills run, with the robots new 8 point stacking capabilities

STRATEGIES FOR ROBORUMBLE

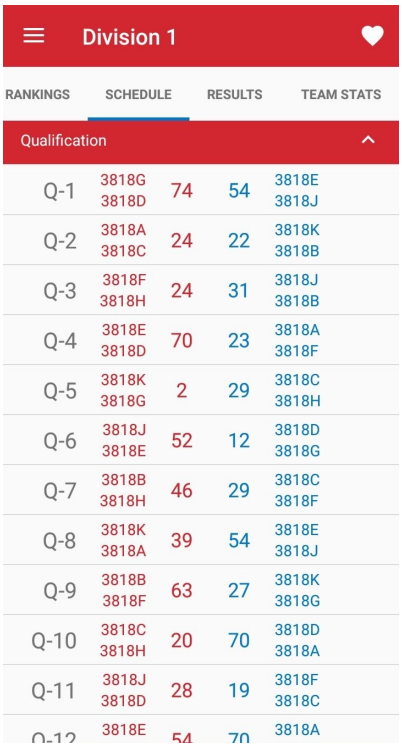
Summary:

1. Scouting teams and capabilities

2. Strategy on the field (Qualification Matches)
3. Strategy during skills runs
4. Maintenance and workflow

As a team, we **discussed the important strategies and things we needed to consider for RoboRumble**. It is important to dive in with a plan in mind so that we don't get lost during the competition.

One major factor we focused heavily on during planning is our strategies with **scouting other teams and potential alliance members**. Although we've done team analyses after Trial of the Pushbots and Thunderdome, teams have significantly improved since then and we'd have to take into account more recent abilities. Apart from this, these analyses only take into account ISM teams, and do not include the 10 other participating in the competition. Seba was given the role of scouter during the event, and he would be in charge of taking note of teams abilities and capabilities from matches and skills, as well as seeing who we'll be competing with and against. One important tool we'd be using is the Vex Via app, which gives us access to all information of the tourney, from match schedule and alliance teams with and against, plus skills rankings and more. An example from last year's Roborumble is seen below.



Q-1	3818G 3818D	74	54	3818E 3818J
Q-2	3818A 3818C	24	22	3818K 3818B
Q-3	3818F 3818H	24	31	3818J 3818B
Q-4	3818E 3818D	70	23	3818A 3818F
Q-5	3818K 3818G	2	29	3818C 3818H
Q-6	3818J 3818E	52	12	3818D 3818G
Q-7	3818B 3818H	46	29	3818C 3818F
Q-8	3818K 3818A	39	54	3818E 3818J
Q-9	3818B 3818F	63	27	3818K 3818G
Q-10	3818C 3818H	20	70	3818D 3818A
Q-11	3818J 3818D	28	19	3818F 3818C
Q-12	3818E	54	70	3818A

We've compiled all teams competing in a spreadsheet document in order to rank their abilities in certain aspects, including driving speed/skill, cap flipping speed, high cap scoring, high flag scoring, parking, auton, and skills ranking. These would be filled on the day, and would be useful for taking a holistic view of who'll be competing.

Team No.	Team Name	(1-10) Drive Speed/Skill	(1-10) Stacking Speed	(1-10) Tower Scoring	(1-10) Defense	(1-10) Auton	Total	Rank	Skills Ranking
3818 B	EDSA						0	1	0
3818 C	Cream						0	1	0
3818 D	Johnny 5 Extension						0	1	0
3818 E	Athena						0	1	0
3818 F	Munchlax						0	1	0
3818 G	Euphoria						0	1	0
3818 H	NAIA na lang						0	1	0
4253 A	Raid Zero A						0	1	0
4253 B	Raid Zero B						0	1	0
4253 D	Raid Zero D						0	1	0
1492 Z	WASABI						0	1	0
51947 A	Robot Lives						0	1	0
51947 B	Ipis						0	1	0
76257 A	Harrow A						0	1	0
76257 B	Harrow B						0	1	0
76257 C	Harrow C						0	1	0
76257 D	Harrow D						0	1	0

We've also put **emphasis on our strategy on the field during qualification matches**. From experience in Thunderdome, we've learned that we should not be reliant on our alliance robot as much as possible, and that we should be functional and competitive even as a singular robot. However, collaborating with the other team to assign roles in scoring stacks and scoring towers will enable us to optimize the total score. We will need to focus on blocks on both sides of the field to score highly, and the duration of each match allows for this. With Eion as our match coach, he would direct us through our strategy in the duration of the match

For skills runs, we would also have to **carefully balance scoring stacks in goals and blocks on towers.** Unlike the matches, we'd likely stay on one side of the field due to the limited time constraints. We would also be able to adapt and make changes for skills in real time, through using the VRC Hub App for score calculation and timing. These interfaces can be seen below.

We've also **organised out our maintenance and workflow** to be followed during the day. We wanted to make sure we prepared for everything that we could think of, and that a simple mistake wouldn't cause some match affecting outcome. We've created a team checklist to follow during the day so that we wouldn't forget any essentials before, during, and after matches. This checklist can be seen below. Apart from this checklist, we've already begun collecting and gathering any spare parts and components needed during the competition.

MAD MAX CHECKLIST

Before a match:

- Check Voltage of Batteries
    - Should be at 50% full
- Autonomous
    - Is it set to the right color/program?
- Match number and time
- Teams with and against
    - Talk with Alliance team beforehand
    - Evaluate opposing alliance beforehand
- Strategy

- General check up of robot
    - Make sure nothing is out of the ordinary

<u>During a match:</u>

- Turn on robot beforehand
- Stick to strategy
- Any notable changes/issues
- Ways to improve robot in the future

<u>After a match:</u>

- Check damages if any
    - Repair if necessary
- When is the next match?
- Is there time to do a skills run?
- Refer back to “Before a match”

ROBORUMBLE (VRC PH NATIONAL CHAMPIONSHIPS)

Summary:

1. Tournament Champions
    a. 2nd place in matches, 3rd in skills – Qualified for Vex Formosa
2. Design Award
3. Analysis of performance during matches
4. Post tournament reflections, observations, and improvements
5. Team Analysis

RoboRumble is the determining eliminations within ISM where 3 ISM teams advance to Vex Formosa in Taiwan. As an official Vex Robotics Competition, it was the Philippine National Championships for Tower Takeover. RoboRumble exposed our team to the style of competition with 7 competing teams outside of ISM, allowing us to be more exposed to cooperation and schedules within Vex Robotics Competitions, especially for the upcoming one in Taipei.

Our team won the tournament. We were the 2nd highest seed, winning 6 of our matches and losing two 2. **We had the highest scoring match** in the entire event, with a total of **43 points**, paired with Harrow D (76275 D). The nature of matches made each one varied based on who we were with and against. The results of these are displayed below.

Qual. #2	10:37:00am Fri	Main Field	4253B	3818H	3818A	3818D	1	19	✔
Qual. #7	11:12:00am Fri	Main Field	3818A	51974A	3818E	3818B	9	6	✔
Qual. #11	11:40:00am Fri	Main Field	3818G	1492Z	3818A	76275A	6	14	✔
Qual. #16	01:14:00pm Fri	Main Field	3818A	3818D	76275B	3818F	17	29	✘
Qual. #22	01:56:00pm Fri	Main Field	51974A	3818A	51974B	3818C	30	26	✔
Qual. #29	08:00:00am Sat	Main Field	3818A	76275C	2521A	3818C	7	15	✘
Qual. #31	08:14:00am Sat	Main Field	4253D	4253A	3818A	76275D	18	43	✔
Qual. #36	08:49:00am Sat	Main Field	76275C	2521A	3818H	3818A	3	34	✔

Afterwards, we were chosen Munchlax, the first seed team, to **form the first seed alliance.** We determined that this was the best path to the finals given our circumstances. Some scouting was done, and we were in a position to choose a team member. In this case, we were able to more closely examine how well we performed in the matches. Our elimination matches are as follows:

QF #1-1	3818F	3818A	2521A	76275A	20	18	✔
QF #1-2	3818F	3818A	2521A	76275A	25	22	✔
SF #1-1	3818F	3818A	3818D	3818G	26	24	✔
SF #1-2	3818F	3818A	3818D	3818G	32	12	✔
Final #1	3818F	3818A	3818C	3818E	23	14	✔
Final #2	3818F	3818A	3818C	3818E	38	15	✔

NB1. It should be noted that these matches were played in a Best of 3 (BO3), where an alliance would have to win 2 matches in order to advance.
We experienced the greatest number of problems during the first few rounds, as parts of the code needed to be re-written due to a bug which froze the position of the tray. This was later fixed for the latter rounds.

We placed 3rd in skills rankings, with 18 points in driver and 1 in autonomous (19 total). As planned previously, we carried out our routines for skills to the best of our ability. Although our autonomous had the ability to score 10 points (as tested before the break), these did not work in our favor on the actual day. John's absence also hindered the testing and quality control of our auton during the week. The analysis and results of each skill run can be seen below.

Skills Type	Score	Scoring Method	Limitations/ Observations	Improvements
Driver	4	• 1 cube on goal zone	–Tray got stuck on goal zone, couldn't place stack well	–Moving the bottom of the tray higher –More practice driving

Driver	18	• 6 cubes on goal zone • 3 cubes on towers	–Had trouble intaking cubes –Resorted to 1 stack and placing on towers	–Experimenting with and fixing the intake, more calibration
Auton	1	• 1 cube on goal zone	–5 pt auton used, but the stack fell after it was scored	–More calibration of code, testing
Auton	0	• ———	–The tray failed to flip backwards, and the robot didn't intake any cubes	–Using the checklist– double checking that the robot is good to go for skills

Apart from this, we were the **recipient of the Design Award**. Our team was chosen for interview with a judge, where we diligently answered questions about our design process and methods when building.

TEAM ANALYSIS #3

Only 3 ISM teams advance from Roborumble to Vex Formosa, us being one of them. Unfortunately, the rest of the teams from ISM were eliminated. Thus, this team analysis focuses on both competing ISM teams to Formosa, as well as Taipei American School teams that we would compete with again at Taiwan. Also bolded are the awards received by each team.

Teams highlighted in green represent ISM Teams competing in Taipei.

Rank	Team	Match Rank	Skills Rank	Strengths	Weaknesses	Notes
1	3818 A Mad Max	2	3	**8 pt stacking**	**Auton needs work**	**Tournament Champions** **Design Award**
1	3818 C Cream	3	1	8 pt stacking Decent driver	Robot is sometimes unstable Auton inconsistent	Excellence Award Robot Skills Champion
3	3818 F Munchlax	1	4	Fairly consistent auton Consistent 6 pt stacking		Tournament Champions

Goals for Vex Formosa 2019

Post tournament reflection points and improvements:

1. Raise the bottom of the tray, to prevent getting stuck on the goal zone.
2. Making all screws on the tray as short as possible to prevent them from clipping the arm when scoring. Replace keps nuts with nylocks so they're locked in place
3. Fixing the codes, especially after John's hiatus. Reducing motor speeds and fixing the issue with the tray getting stuck on an angle
4. Adding an anti-tip mechanism to keep the robot from falling over
5. Change the gear ratio of the motors of the intake to increase the speed of intaking cubes
6. Use Polycarbonate for the tray of the robot in order to reduce the amount of friction faced by the intake motors when intaking cubes

Strats/Considerations:

1. Having a “safety auton” for skills, one that can score the regular 8 cube stack without being interfered by the
    a. Shooting 1 high flag, and parking on center (8 points)
    b. Shooting 2 flags, and parking on center (10 points)
2. More practice driving, especially with skills. A lot of unnecessary movements and getting stuck on blocks. Work on strategies on the field.
3. More coordination with alliances beforehand, emphasis on scouting techniques with paired match ups
4. A more systematic checklist to include things that were not taken into account for. Many times, the checklist was not even brought to the match, and as a result, slip ups such as plugging all batteries or setting the auton were overlooked.

As a result of this, we set ourselves a timeline of goals to accomplish to keep us on track for Vex Formosa.

Target Date	Goals	Date achieved / notes
November 16	Finish with clean up from RoboRumble and general maintenance of the robot	finished on November 15
November 21	5 point auton is consistent Code is fixed after John's hiatus	somewhat consistent 5 point auton and consistent 30 second 8 point auton on November 21
November 23	Structural components must be 95% complete, 8 pt stacking should be consistent	achieved on November 23, although some structural components need tweaking
November 26	At least 30 point driver skills again	42 point skills achieved on November 21
November 29	Consistent 5 pt for matches, and at least 10 pt skills auton. At least 40 pt skills driver	Consistent 5 pt auton, 48 point driver skills achieved on November 23
December 2	Consistent 8 pt auton for matches	achieved on December 4, delays due to typhoon

November 11 and 13, 2019 (Eion)

Summary:

1.) General Cleanup
2.) Disassembling eliminated robots

These sessions' main foci were basically cleaning up. After the Roborumble tournament, we needed to return most of the equipment back to the robotics room, which we promptly cleaned up afterwards. In addition, we had the task of disassembling the eliminated robots. While doing so, we made sure to lookout for potential mistakes that we may make or potential innovative ideas that we may

incorporate into our robot. Unfortunately, many of the ideas were not compatible with our robot since it is already right below the size limit.

November 16, 2019 (Jason)

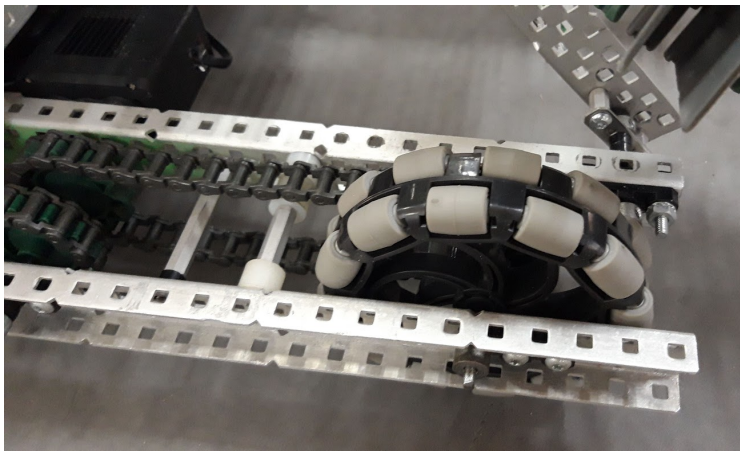
Summary:

1. Saturday Service
2. Moved Tray up

The majority of today was spent teaching Vex robotics to kids from a local elementary school. As a result, we had much less time today in working on the robot.

The primary thing that we were able to achieve was moving the lower axle of the tray up. This made it such that the bottom of the tray did not clip with the goal zone barriers anymore. Thus, this made scoring stacks much more efficient.

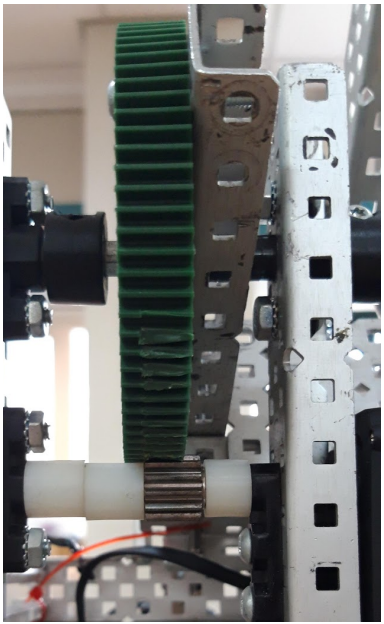
November 19, 2019 (Eion)



Summary:

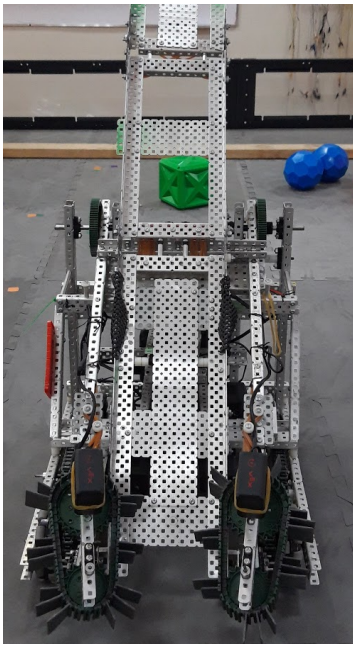
1. Changed wheels
2. Reversed c-channel base
3. Changed gears

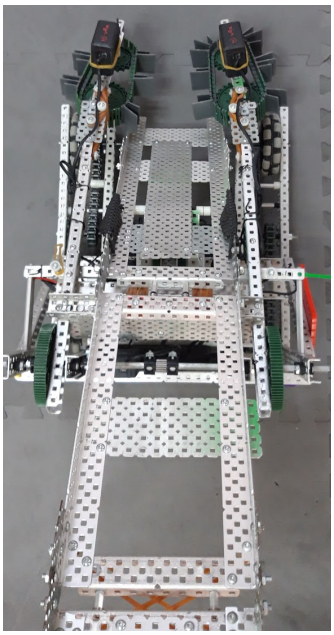
As seen in the picture, we replaced the wheels with new ones since the old ones were stripping already. We did this to ensure the robot's stability throughout the match, thus reducing the possibility of breaking or malfunction. The replacement we used were the new gray wheels. Additionally, we also reversed the c-channel base. We initially tried to reverse it in order to attach an anti-tipping mechanism to the back of the robot. However, we deemed this unnecessary due to its resultant misalignment and thus destabilization to the robot. Another design for the anti-tipping



must be thought of. This would reduce the time wastage when the robot accelerates too quickly, as currently, one needs to slow it down first in order to prevent the robot from tipping backwards. Moreover, it would prevent the unlikely, yet potentially disastrous, possibility of the robot tipping over and being inactive the rest of the match. We again changed the gears to make the front gear slightly smaller than the rear gear. We tested both possibilities and this seems to be, by far, the better option.

We also addressed an issue with the arm, which started to struggle with lifting after Roborumble. We found that the high strength 12-tooth gear was misaligned with the 84-tooth gear raising the arm. This caused the large gear to begin deforming in its teeth, as seen on the right. To address this, we added spacers to re-align both gears. Afterwards, the arm raised substantially better.





November 20, 2019 (John)

Summary:

1. Program Explanation

This will be the main section that explains the progression of the code so far. The program mainly consists of the following divisions:

1. **Usercontrol code**
2. **Autonomous Code**
    a. **5 points**
    b. **8 points**
3. **Programming Skills**

Before proceeding, below is the list of all the functions that are custom defined. They improve program concision and make the code easier to understand as well.

```
//--------------Global Variables----------------
double armAngle = 0; //deprecated, used in old versions
int intakeState = 0;
int manual = 0; //deprecated, used in old versions
int countr = 0;
int y = 0; //deprecated, used in old versions
int rampState = 0;
//----------------------------------------------
//callback function used later. Switches between manual and automatic control for the ramp
void rampMode() {
 if(rampState == 0) {
   rampState = 1;
 } else if(rampState == 1) {
   rampState = 0;
 }
}
void velocitySet(double left = 30, double right = 30) { //sets the velocity of the drivetrains
 LeftFront.setVelocity(left,vex::percentUnits::pct);
 LeftBack.setVelocity(left,vex::percentUnits::pct);
 RightFront.setVelocity(right,vex::percentUnits::pct);
 RightBack.setVelocity(right,vex::percentUnits::pct);
}
double todeg(double cm) { //converts a distance in centimeters to degrees of wheel rotations
 double wheelDiameter = 10.16;
 double circumference = wheelDiameter * 3.141592;
 double degreesToRotate = 360 * cm/circumference;
 return degreesToRotate;
}
void forwardDistance(double cm, double vel) { //moves forward(or backward if cm<0) a certain distance
```

```
 velocitySet(vel,vel);
 double deg = todeg(cm);
 LeftFront.rotateFor(deg, vex::rotationUnits::deg, false);
 LeftBack.rotateFor(deg, vex::rotationUnits::deg, false);
 RightFront.rotateFor(deg, vex::rotationUnits::deg, false);
 RightBack.rotateFor(deg, vex::rotationUnits::deg);
}
void forwardTime(double s, double vel) { //moves forward(or backward if cm<0) for a certain time
 velocitySet(vel,vel);
 LeftFront.spin(vex::directionType::fwd);
 LeftBack.spin(vex::directionType::fwd);
 RightFront.spin(vex::directionType::fwd);
 RightBack.spin(vex::directionType::fwd);
 vex::task::sleep(s);
 LeftFront.stop();
 RightFront.stop();
 LeftBack.stop();
 RightBack.stop();
}
void turn(bool directn, double degrees, double vel) { //makes the robot turn, “direction” is a used name
 velocitySet(vel,vel);
 if(directn == 0) { // turns right
   LeftFront.rotateFor(degrees, vex::rotationUnits::deg, false);
   LeftBack.rotateFor(degrees, vex::rotationUnits::deg, false);
   RightFront.rotateFor(-degrees, vex::rotationUnits::deg, false);
   RightBack.rotateFor(-degrees, vex::rotationUnits::deg);
 } else {//turns left
   LeftFront.rotateFor(-degrees, vex::rotationUnits::deg, false);
   LeftBack.rotateFor(-degrees, vex::rotationUnits::deg, false);
   RightFront.rotateFor(degrees, vex::rotationUnits::deg, false);
   RightBack.rotateFor(degrees, vex::rotationUnits::deg);
 }
}
```

Usercontrol

The main changes involve improvements to the arm and ramp mechanism whereas the base retains the tank control discussed earlier. The following includes all relevant programs that relate to th ramp and arm mechanism. The explanations are detailed in the comments(green font)

```
void arm() {
 /*
 These commands set up the initial values for the robot.
 The "setBrake(brakeType::hold)" commands for the ramp and armLift
 will make it so that executing a .stop() function will hold the motors in place(using power).
 The ramp and armLift are designated with 45% and 100% power respectively
 /**/
 ramp.setBrake(brakeType::hold);
 armLift.setBrake(brakeType::hold);
 ramp.setVelocity(45,vex::percentUnits::pct);
 armLift.setVelocity(100,vex::percentUnits::pct);
 //------------manual mode---------------//
 /*
 This is the manual mode. Modes are changed by the callback function rampMode. The robot initially
 starts in automatic mode but the button Y on the controller switches between the two modes.
 As the name suggests, in this mode all of the mechanisms are controlled manually.
 /**/
 if(rampState == 1) {
   controller1.Screen.clearScreen();
   controller1.Screen.print("Manual Mode");
   if(controller1.ButtonUp.pressing()) {
     ramp.spin(vex::directionType::fwd); //countr allows this call
   } else if(controller1.ButtonDown.pressing()) {
     ramp.spin(vex::directionType::rev); //and this call to function properly
   } else {
     /*
     This bit of code is the only "automatic" part in manual mode. It is a remnant from the previous
     version of the code when the modes were combined(outlined previously). This is just here in case
     the driver forgets to return to automatic mode whenever he lifts the arms as the ramp is usually
     in the way. However, there is a flaw in the driver must wait about 1-2 seconds for the ramp to extend
     as this was the only option for the previous iteration's combined set of functions. The problem
     disappears entirely in automatic mode.
     /**/
     if(armLift.rotation(vex::rotationUnits::deg) > 50) {
       countr = 1;
       /*
       "countr" sets a boundary at 50º of rotation. Because an infinite loop is being run, countr
       marks when the arm is raised and tells the ramp to incline a bit to move out of the way.
       /**/
       ramp.rotateTo(350,vex::rotationUnits::deg,false);
     } else if(armLift.rotation(vex::rotationUnits::deg) < 50 && countr == 1) {
       /*
       Here the logic detects when the arm is lowered(<50º). If this is true, then the robot
       knows that the arm is lowered and that the ramp can be returned to its original position
```

```
      without getting in the way of the arm and vice versa. Countr is reset to 0.

      Countr redirects the ramp.stop() command, allowing ramp.spin() to allow manual control and
      preventing the code immediately following this comment from running when the arm isn't raised
      initially-thus lowering the ramp regardless of whether the user wants to or not. However,
      this is also the reason why the robot must wait for ramp.rotateTo(10,vex::rotationUnits::deg);
      to finish, as if the logic simply went through, then countr would be 0 and also ramp.stop() would
      prevent the command from running.
      /**/
      ramp.rotateTo(10,vex::rotationUnits::deg);
      countr = 0;
    } else {
      ramp.stop();
    }
    ramp.stop();//didn't get time to check but probably unnecessary
  }

  /*
  These are manual commands for lifting the arm. the .setbrake() call solves the issue of
  the motors slowly falling due to weight.
  */
    if(controller1.ButtonR1.pressing()) {
    armLift.spin(vex::directionType::fwd);
  } else if(controller1.ButtonR2.pressing()) {
    armLift.spin(vex::directionType::rev);
  } else {
    armLift.stop();
  }
 }
//------------automatic mode---------------
/*
This is the automatic mode. This is the default mode when the code is run. In this mode,
the motors(except for the drivetrains) are set to rotate a specific amount, specifically:

ramp: perpendicular and declined position
armLift: rest, height for lowest tower, and height for middle-height tower(highest tower not physically
attainable)
/**/
 else {
  //this section controls the movement of the ramp
  controller1.Screen.clearScreen();
  controller1.Screen.print("Automatic Mode");
  if(controller1.ButtonUp.pressing()) {
    ramp.rotateTo(1300,vex::rotationUnits::deg,false); //perpendicular position(stacking)
  } else if(controller1.ButtonDown.pressing()) {
```

```
    ramp.rotateTo(10,vex::rotationUnits::deg,false); //declined position(intake / rest)
  } else {
    //This controls the ramp when the arm is raised
    //This is similar to the code mentioned above. However,..
    if(armLift.rotation(vex::rotationUnits::deg) > 30) {
      countr = 1;
      ramp.rotateTo(500,vex::rotationUnits::deg,false);
    } else if(armLift.rotation(vex::rotationUnits::deg) < 30 && countr == 1) {
      ramp.rotateTo(10,vex::rotationUnits::deg,false);
      /* ... notice that the false tag is now applicable. This is because there are
      no ramp.stop() calls anywhere in automatic mode as they are all set degrees
      of rotation
      /**/
    }
  }
//This section controls the raising of the arm
  if(controller1.ButtonX.pressing()) {
    armLift.rotateTo(1000,vex::rotationUnits::deg,false); // middle-height tower
  } else if(controller1.ButtonA.pressing()) {
    armLift.rotateTo(760,vex::rotationUnits::deg,false); // lowest tower
  } else if(controller1.ButtonB.pressing()) {
    armLift.rotateTo(10,vex::rotationUnits::deg,false); // rest
  }
}


//These functions run regardless of whether in automatic or manual mode
if(controller1.ButtonL1.pressing()) { // makes the rollers intake blocks
  armLeft.setVelocity(100,vex::percentUnits::pct);
  armRight.setVelocity(100,vex::percentUnits::pct);
  armLeft.spin(vex::directionType::fwd);
  armRight.spin(vex::directionType::fwd);
} else if(controller1.ButtonL2.pressing()) { //spins the rollers in reverse / outtake
  armLeft.setVelocity(100,vex::percentUnits::pct);
  armRight.setVelocity(100,vex::percentUnits::pct);
  armLeft.spin(vex::directionType::rev);
  armRight.spin(vex::directionType::rev);
} else {
  /*
  intakeState is changed through a callback function "intakeMode". When it is true or 1, the
  rollers are set to spin automatically at 100% velocity to intake(though overridable by the
  controller commands) and when it is 0(default) the rollers simply stop moving when nothing
  is pressed
  /**/
  if(intakeState == 0) {
```

```
      armLeft.stop();
      armRight.stop();
    } else {
      armLeft.spin(vex::directionType::fwd,100,vex::velocityUnits::pct);
      armRight.spin(vex::directionType::fwd,100,vex::velocityUnits::pct);
    }
  }
}
```

The following are the declarations and the calls of the mentioned callback functions rampMode and intakeState

Declarations(global scope)	Calls(within *void usercontrol()*)

`void rampMode() {`
` if(rampState == 0) {`
`  rampState = 1;`
` } else if(rampState == 1) {`
`  rampState = 0;`
` }`
`}`
`void intakeMode() {`
` if(intakeState == 0) {`
`  intakeState = 1;`
` } else if(intakeState == 1) {`
`  rampState = 0;`
` }`
`}`
	`void usercontrol() {`
	` controller1.ButtonY.pressed(rampMode);`
	` controller1.ButtonRight.pressed(intakeMode);`
	` while(true) {`
	`  arm();`
	`  tank();`
	`  vex::task::sleep(10);`
	` }`
	`}`

Programming Skills

Because the focus is on the autonomous code for matches, there was less focus on the skills programming. However, due to the rules, one cube in the tower doubles the score of all the stacked cubes and thus I extended the match autonomous code and added a portion wherein it stacks at a tower. The main differences are:

1. Slower intake and overall drivetrain speed(~20% drivetrain power)
2. Additional “safety mechanism” during the stacking process to greatly improve consistency–guarantees **at least 8 points most of the time**

   [Appended Section]

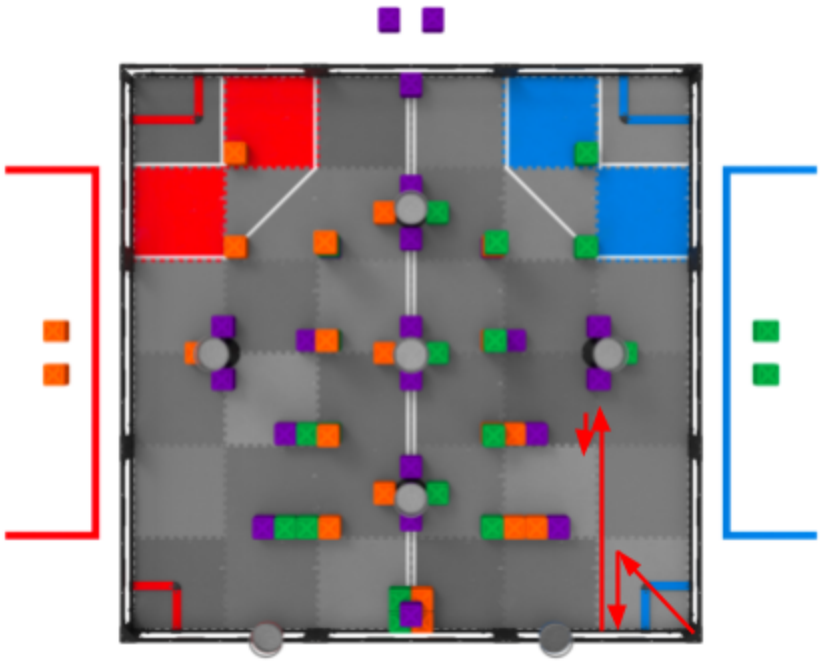
```
forwardTime(2500,50);
vex::task::sleep(500);
forwardTime(500,-10);
```

```
armLeft.setVelocity(25,vex::percentUnits::pct);
armRight.setVelocity(25,vex::percentUnits::pct);
armLeft.rotateFor(-50,vex::rotationUnits::deg,false);
armRight.rotateFor(-50,vex::rotationUnits::deg);
vex::task::sleep(500);
armLeft.setVelocity(20,vex::percentUnits::pct);
armRight.setVelocity(20,vex::percentUnits::pct);
armLeft.spin(vex::directionType::fwd);
armRight.spin(vex::directionType::fwd);

ramp.setVelocity(30,vex::percentUnits::pct);
ramp.rotateTo(1300,vex::rotationUnits::deg);
vex::task::sleep(500);
forwardTime(800,10);
vex::task::sleep(500);
armLeft.setVelocity(-15,vex::percentUnits::pct);
armRight.setVelocity(-15,vex::percentUnits::pct);
armLeft.rotateFor(-800,vex::rotationUnits::deg,false);
armRight.rotateFor(-800,vex::rotationUnits::deg,false);

forwardDistance(-45,30);
armLeft.stop();
armRight.stop();
```

The portion shaded above in a darker color represents the key addition. The part before the 500ms pause pushes the stack down by a bit. Our robot doesn't really let the blocks simply slide off when the ramp is raised so this allows the stack to not be dropped onto the floor and tumble in the later part of the code. The part after the 500ms pause solves the issue where the bottom block is usually not aligned with the rest of the stack and usually causes imbalances. To solve this, the bottom block is dragged back into alignment using the rollers. This works in 8 point stacks as the weight(when stacking) pushes the block down and the ramp also aligns all the blocks

.

3. Extended code to stack on one tower to attain **16 points**
   The following code is added to the end of the slowed-down autonomous code. The details are explained in the comments(green font) and the path of the robot is shown below

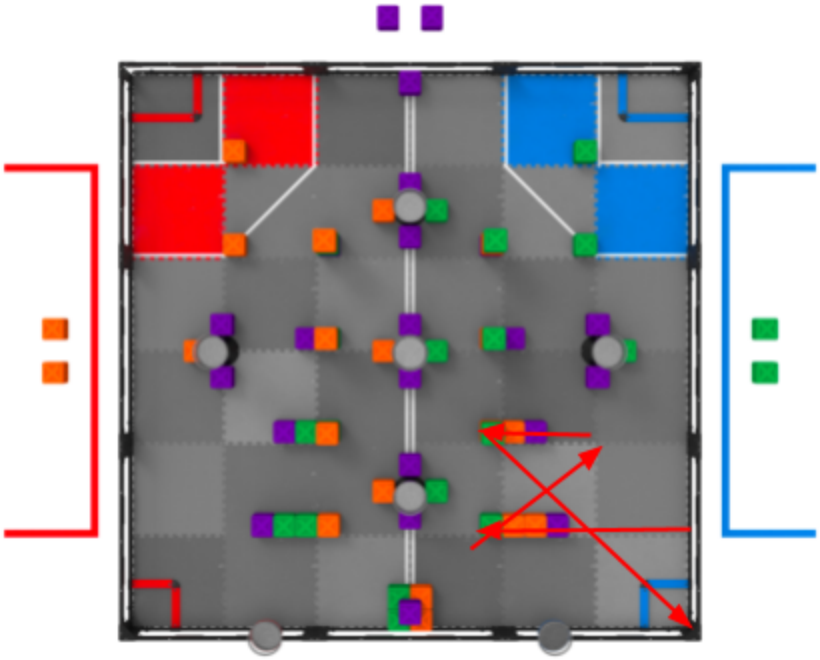


```
forwardDistance(-45,30);
armLeft.stop();
armRight.stop(); //^^^ from last snippet

turn(1,430,15);   // rotates the robot so that it's back is facing the wall
vex::task::sleep(500);
ramp.rotateTo(400,vex::rotationUnits::deg,false); // mid-declines the ramp to enable arm lifting
forwardTime(2500,-50); //moves the robot to the wall, aligning itself to the tower
vex::task::sleep(500);

forwardDistance(102,20); //moves the robot 102cm, just enough to intake the violet cube on the tower
vex::task::sleep(500);
armLeft.setVelocity(40,vex::percentUnits::pct);
armRight.setVelocity(40,vex::percentUnits::pct);
armLeft.rotateFor(500,vex::rotationUnits::deg,false);
armRight.rotateFor(500,vex::rotationUnits::deg); //intakes the cube to be in between the rollers

vex::task::sleep(500);
forwardDistance(-10,20); //moves backward 10cm to avoid hitting the tower accidentally
vex::task::sleep(500);

armLift.setVelocity(50,vex::percentUnits::pct);
armLift.rotateTo(900,vex::rotationUnits::deg); //raises the arm
vex::task::sleep(200);
forwardDistance(30,20); //moves forward in preparation to put the block into the zone
vex::task::sleep(500);
armLeft.setVelocity(-100,vex::percentUnits::pct);
```

```
armRight.setVelocity(-100,vex::percentUnits::pct);
armLeft.spin(vex::directionType::fwd);
armRight.spin(vex::directionType::fwd); //outtakes the block, hopefully it lands in the tower
vex::task::sleep(1000);
forwardDistance(-80,30); //makes the robot go backward
armLift.stop();
armLeft.stop();
armRight.stop(); //all motors stop, program ends.
```

Autonomous Code(for matches)

The autonomous code is split into mainly 3 parts. These are: the **expansion**, the **intake**, and the **stacking**. The **expansion** bit is simple. It simply involves turning the rollers for half a second to expand from size-restricted mode and then reversing their direction(so that it intakes). The robot is then realigned by making it go against the wall behind it. The **intake** process involves making the robot following the path depicted in the image below. Note that when it reaches the 8th block, the intake is slowed down to prevent overflow but still keep the blocks in place.



```
void bluEight() {
 //------------Expansion--------------
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::rev);
 armRight.spin(vex::directionType::rev);
 vex::task::sleep(500);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
 forwardTime(300,-40);
 vex::task::sleep(150);
 //----------------intake--------------
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);//
 armRight.spin(vex::directionType::fwd);

 forwardDistance(85,35);
 vex::task::sleep(200);
 turn(1,130,25);
 vex::task::sleep(200);
 forwardDistance(-80,40);
 vex::task::sleep(250);
 turn(0,130,25);
 vex::task::sleep(250);
 forwardDistance(70,30);

 armLeft.setBrake(hold);
 armRight.setBrake(hold);

 armLeft.setVelocity(15,vex::percentUnits::pct);
 armRight.setVelocity(15,vex::percentUnits::pct);
  turn(1,390,30);
 vex::task::sleep(300);
 //------------------stacking-------------------
 ramp.setVelocity(25,vex::percentUnits::pct);
 ramp.rotateTo(1300,vex::rotationUnits::deg,false);
 forwardTime(2500,50);
 forwardTime(400,-10);
 vex::task::sleep(100);
 ramp.setVelocity(50,vex::percentUnits::pct); //ramp speeds up to keep up with time
 armLeft.setVelocity(25,vex::percentUnits::pct);
 armRight.setVelocity(25,vex::percentUnits::pct);
```

```
armLeft.rotateFor(-70,vex::rotationUnits::deg,false);
armRight.rotateFor(-70,vex::rotationUnits::deg);
vex::task::sleep(100);
forwardTime(400,10); //the ramp will be perpendicular by the time this program runs
vex::task::sleep(100);
armLeft.setVelocity(-15,vex::percentUnits::pct);
armRight.setVelocity(-15,vex::percentUnits::pct);
armLeft.rotateFor(-800,vex::rotationUnits::deg,false);
armLeft.rotateFor(-800,vex::rotationUnits::deg,false);
forwardDistance(-30,40); // 25
armLeft.stop();
armRight.stop();
```

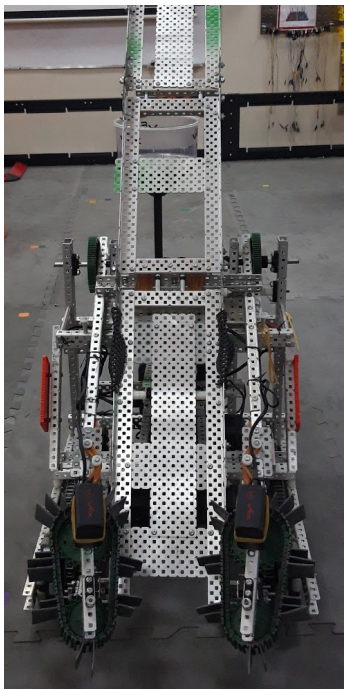
Due to time limitations, the stacking process isn't as complex as the one for programming skills. As soon as the robot heads for the goal, the ramp is set to slowly go perpendicular. There is a section where the ramp speeds up to 50% to keep up with the commands and the time. To at least increase consistency we make the robot go forward for 400ms to secure the blocks in place before making the robot go backward

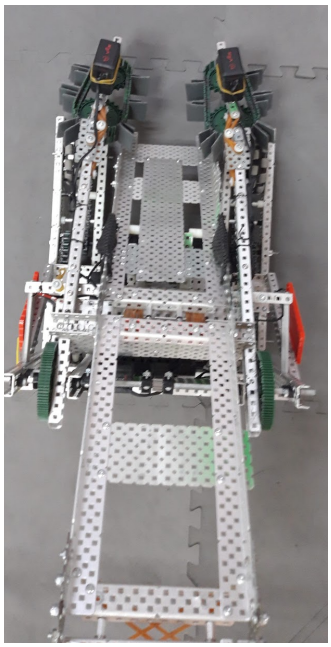
November 21, 2019 (Jason)

Summary and Goals Accomplished:

1. 42 point driver skills run
2. Consistent 8 point skills autonomous
3. Consistent 5 point match autonomous

Ahead of schedule, I was able to do a 42 point skills run by scoring 2 stacks of 7 cubes, and placing 2 cubes on towers. Apart from this, John was able to successfully experiment with the code again, to the point where his 5 point match auton and 8 point skills autons are fairly consistent. General maintenance on the robot was also done today.

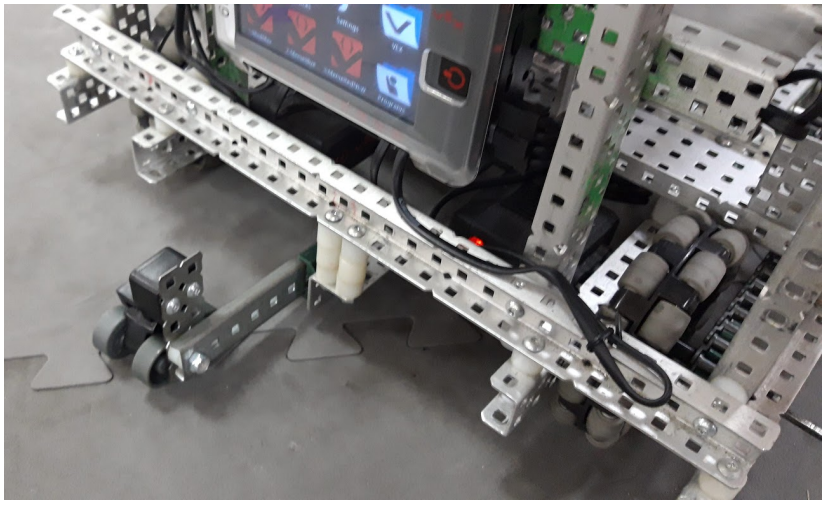




November 22, 2019 (Eion)

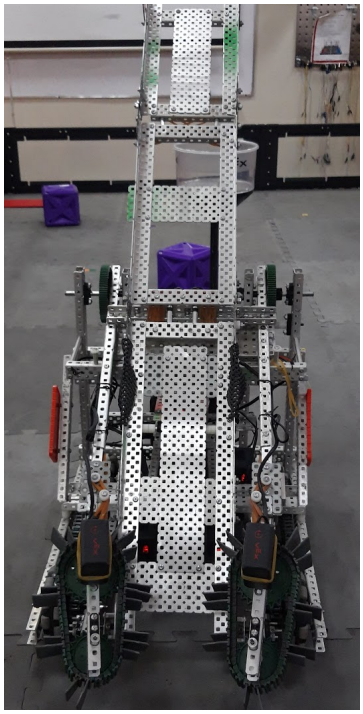
Summary and Goals Accomplished:

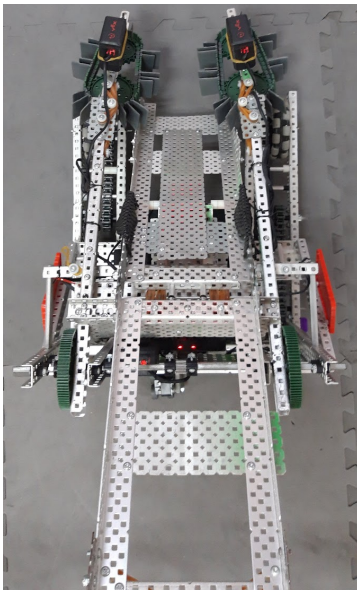
1. Anti-tipping mechanism prototype
2. Driving practice
3. Polycarbonate barriers on the ramp edges



As shown in the image, a prototype anti-tipping mechanism was built. Small wheels were used at the bottom to reduce friction and sudden jarring impacts to the robot. There were numerous challenges in creating this, however, as our robot is very closely packed at the bottom. As such, the battery prevented the slider from going far enough into the robot,

thus causing the mechanism to go out of bounds even when pushed in. Hence, it was therefore necessary to lower the mechanism with either spacers or standoffs. Then, it became necessary to connect the wheels higher up on the slider in order to place it slightly off the ground. Since the battery could not be moved elsewhere easily, the use of positional adjustments was necessitated, which makes the mechanism more and more unstable or bendable. In addition to this prototypal modification, we underwent a lot of driving practice. The main foci were picking up blocks efficiently without wasting too much time in the process and stacking 8 block stacks consistently. We also attempted many two 8 block stacks in the large goal zone. Furthermore, we added two polycarbonate plates near the top of the ramp with the caption “MAD MAX” on it. These plates would serve to stabilize the top block, which often moved around and fell off without these. In addition, they usually destabilized the other blocks when stacking. These plates solved that problem.



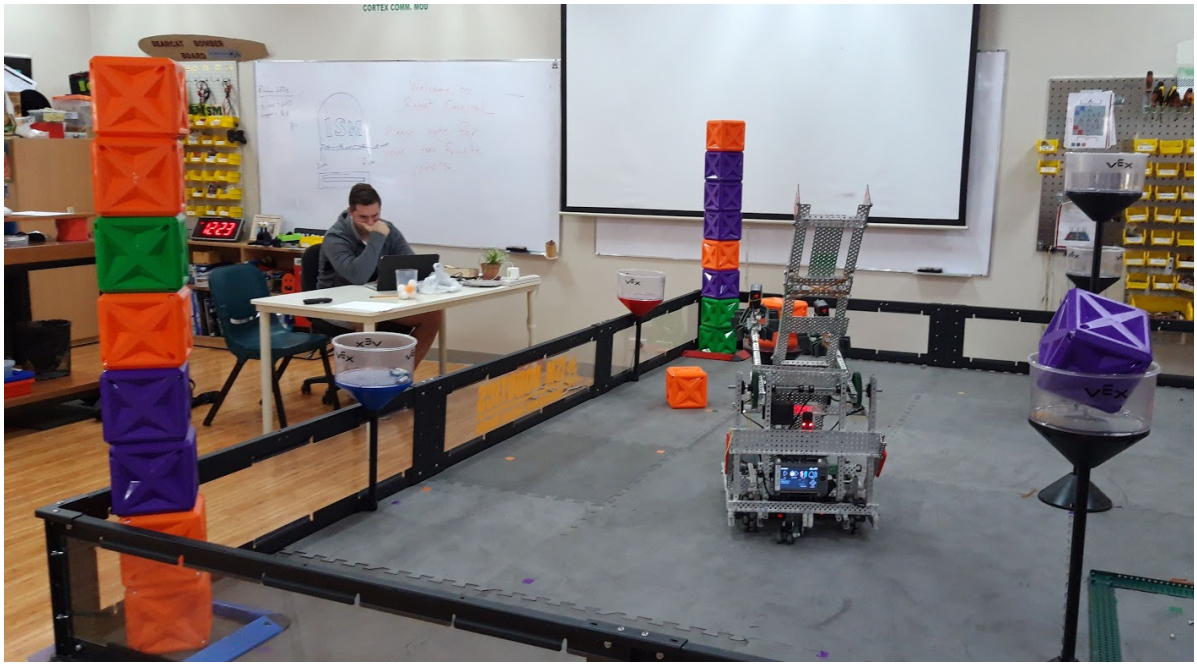


November 23, 2019 (Jason)

Summary and Goals Accomplished:

1. 48 point driver skills

The majority of this session was spent practicing driving skills and aiming to improve from last time, especially trying improve the 42-point attempt. I was able to score 48 points in driver skills. This was done by doing the same routing as last time by stacking 2 8 point stacks on the end goal zones, then placing two cubes on towers. We were not only ahead of schedule in terms of the ability to score points. Now, it is just a matter of practicing more to improve my driving with the robot, both in terms of matches and skills. This session was cut short because of other commitments that everyone else had.



The 48-point driver skills scored, by placing 2 cubes on towers and scoring 2 8-block high stacks

November 25, 2019 (Eion)

Summary and Goals Accomplished:

1.) Practice stacking 8 blocks
2.) Practice placing on towers

This session was mainly devoted to practice driving the robot. One major issue was that the robot was accelerating too quickly at times, which made the robot quite unstable. As such, John edited the driving settings to Jason's preferences. A main point that we focused on here was the consistency in scoring. In a match, failure in even just one stack can result in a huge disadvantage against the opponent. As such, we made sure that the 8 stacks would be very stable, consistent, and efficient. In addition to that, we also focused on placing blocks on both the high and low towers.

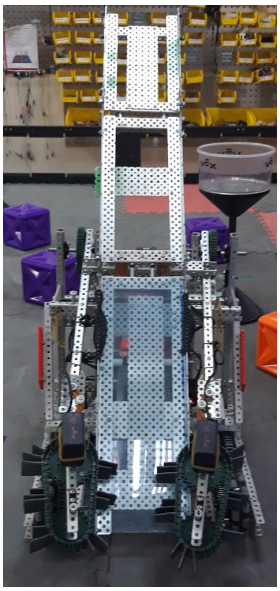
November 26, 2019 (Eion)

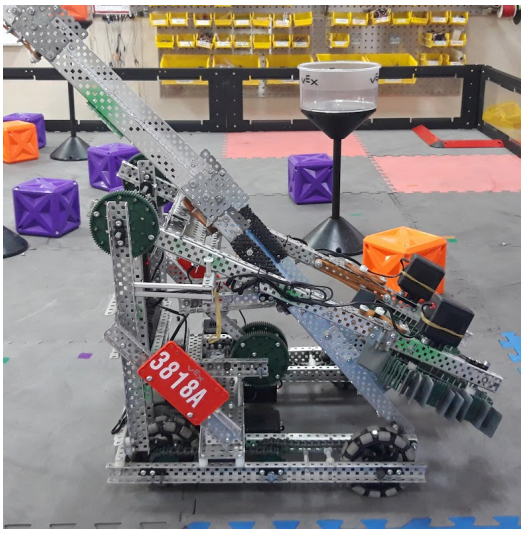
Summary and Goals Accomplished:

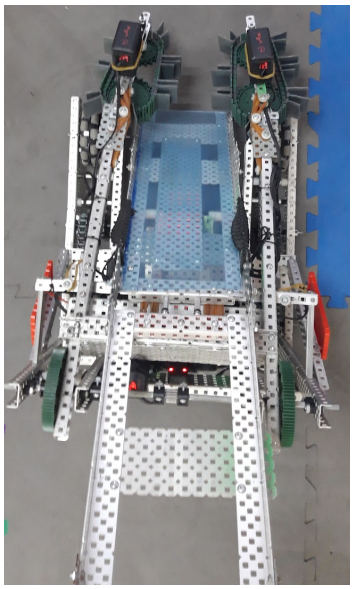
1. Anti-tipping mechanism
2. Polycarbonate ramp testing

We created another design for the anti-tipping mechanism. We decided that the sliders were too much of a risk at this point in time before the competition, so we settled with a slightly protruding steel plate at the back of the robot to prevent it from tilting too much backwards. This replaced the previous prototype of the slider and wheels mechanisms, and it seemed to fit our needs quite well. In addition, we ran a plethora of tests to see whether or not the polycarbonate plating would help the stability of the blocks when on the ramp. We noticed that covering the various irregularities such as the slightly protruding screw heads and the aluminum plates with the polycarbonate truly reduced the amount of bumping about on the ramp itself. It also provided a flat surface for the cubes to rest on, which made them less likely to fall off when the robot accelerated and turned quickly. This also helped align the blocks

more precisely when stacking them onto the goal zones, as they were all uniformly pressed against the ramp face.







November 27, 2019 (Eion)

Summary and Goals Accomplished:

1.) Practice Driving Skills: almost scored 64 points
2.) Increasing driving consistency
3.) Changed motor torques

This session was mostly devoted to practicing the timed driving skills. Being able to practice in a very similar scenario may help our primary driver, Jason, focus during the actual driver skills runs. In addition to that, our secondary driver, Justin, practiced driving the robot as well. During our driving skills run, we almost scored 64 points, which is our goal. Our goal is to stack two stacks of 8 blocks, and then place 3 blocks on towers. This will result in a total driver skills score of 64 points. After these attempts, we again aimed to increase driving consistency in order to increase the reliability during both matches and skills driving. Then, we also changed the motor torques. This is important since although our intake mechanisms could now pull in 8 blocks onto the ramp, its spinning speed is low due to the high torque. As such, it is difficult to finish the autonomous program quickly since the robot needs to travel forwards slowly to

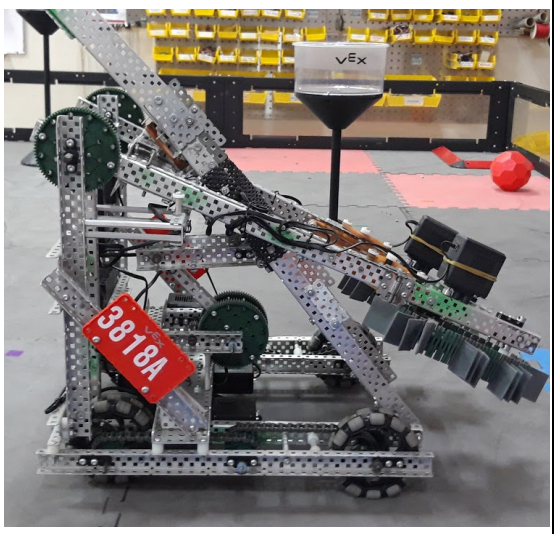
intake the blocks consistently. As such, we changed the motor from the red motor to the green motor in the hopes of being able to intake faster. However, we still needed to be able to intake 8 blocks. As such, we also plan to change the gear ratio of the intake to make the effective torque lower than the red motor but higher than the green motor.

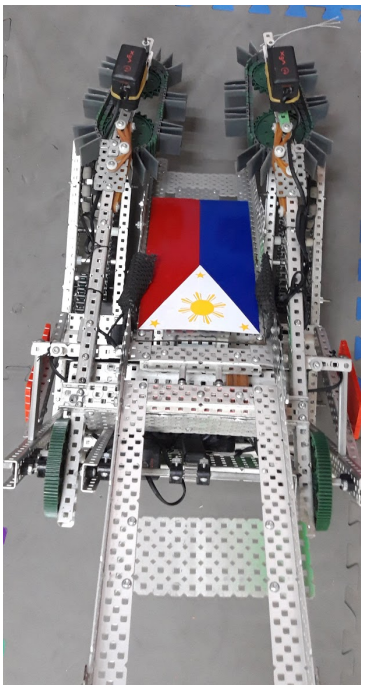
November 29, 2019 (Eion)

Summary and Goals Accomplished:

1. Cutting the Polycarbonate tray piece
2. Vinyl decorations

Today we now replaced the old polycarbonate plate with a new, custom-fitted one. The old one we used was pre-cut and not specifically designed for our dimensions and specifications. We cut the polycarbonate to fit the bottom section of the tray exactly without obstructing any of the other nearby mechanisms. In addition to that, we added vinyl decorations to make our robot more personalized and identifiable. This primarily consisted of the large Philippines flag on the first section of the ramp.





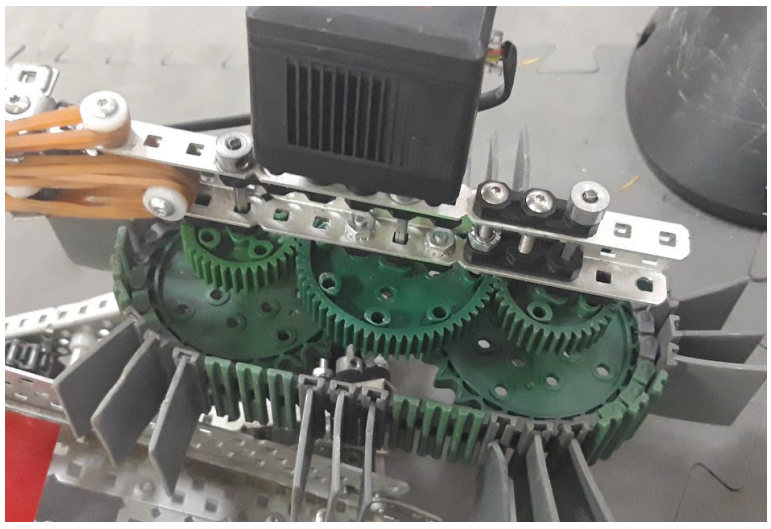
December 2, 2019 (Jason)

Summary and Goals Accomplished:

1. Changed gear ratio of intake
2. Build sessions cancelled due to Typhoon Kammuri

The build sessions that were supposed to occur on Saturday, today and tomorrow were unfortunately lost. It was Bonifacio day on November 30, and as a result we weren't allowed to enter the campus. Further, because of the landfall of Typhoon Kammuri in the Philippines, Metro Manila announced the raising of Typhoon Signal no. 2, cancelling all after school sessions on Monday and Tuesday. As a result, **Taipei Tuneup, which was an opportunity to practice with matches did not pull through.**

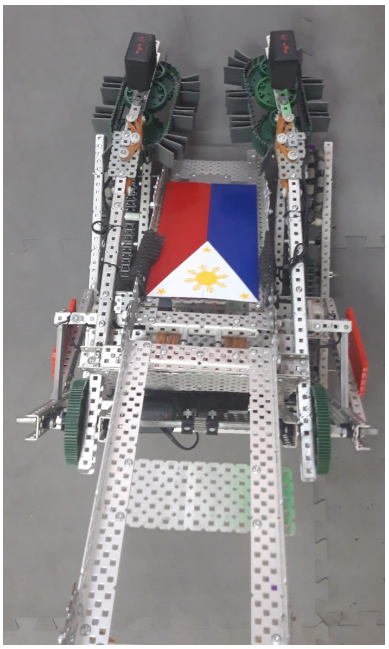
The main thing that I was able to address today during a free period was **increasing the speed of the intake motors.** The main problem that John faced achieving the 8 point 15 second auton was that the intake speed was too slow, thus it took more time to intake and stack the blocks. While we were currently using 100 rpm motors, the dilemma was that 200 rpm was too weak to support more than 5 cubes before stalling, even with ample spacing and tension. Thus, I created a gear ratio beneath the motor that would increase the speed between 100 and 200 rpm. This was achieved by using two 32-tooth gears being driven by a 60-tooth gear connected to the motor. The motor was subsequently moved to the center of the c-channel, and the spacing from before did not need to be changed, an extra delrin bearing was simply added at the center of the original two. This produced a 32:60 gear ratio with a new speed of 187.5 rpm.



After initial testing, some tweaks needed to be made. Originally I used an 18-tooth sprocket connected to a 24-tooth

sprocket. However, this change in diameter twisted the axles and caused the intake to stop working. To address this, I changed the size of the intakes to both have 24-teeth, and this significantly improved the results. After more initial tweaking, the intakes worked properly. I started testing the new intake capabilities and it was still strong enough to support 8 cubes, albeit it gets somewhat slow in the last few cubes. This also caused the motors to start to stall more easily, which was a problem for practicing over extended periods of time, usually when running the robot for over 5 minutes long.

Afterwards, John went in to test the 8 point autonomous he had originally written, which was 20 seconds long. With the minimal time he had testing, he was able to reduce this to 16.5 seconds long. This was a significant improvement, and we all agreed that an advantage gained in an 8 block tall stack in the unprotected zone was more valuable than the possibility of stalling when practicing, as well as the fact that the match wouldn't extended over that long anyways. Because of the changes in schedule, we began to plan over how to make up for the lost time before we leave on Thursday.



STRATEGIES FOR VEX FORMOSA 2019

Summary:

1. Refined maintenance and workflow
2. Strategy on the field (Qualification Matches Analysis)
3. Strategy during skills runs
4. Maintenance and workflow
5. Improvement on scouting teams and capabilities

As a team, we **discussed the important strategies and things we needed to consider for Vex Formosa**. Having learned valuable lessons during RoboRumble, we maintained the emphasis on sticking to a plan and schedule in order to dive into the competition well. Unlike RoboRumble, Formosa does not give us a home advantage. Travelling to Taiwan complicates our access to materials and time spent refining code and driving. With this also being a much bigger competition, the importance of maintaining these strategies is heightened even further.

One major factor we focused on improving for Formosa is our **ability to scout other teams, potential alliance members, and analysis of our upcoming matches**. Unlike at RoboRumble, our team analyses are only limited to ISM teams and TAS teams who've competed at RoboRumble. This increased the uncertainty of who we would be allied with during the 7 qualification matches we have. Luckily, the match schedule was released in advance, and we were able to take a more in depth look at what our schedule would look like prior to the competition. Some things of note:

1. There are a total of 36 teams in the competition
2. We will most likely have 4 practice matches and 7 qualification matches
3. Because of the scale of the event, some qualification matches extend to the second day of the competition
4. Our practice matches sometimes contain competing teams during qualification matches, giving us an opportunity to plan a strategy against them in qualification matches

Apart from this, the nature of travelling to Taiwan brings to light the heightened importance of packing materials, spare parts, and other items we may need during the competition. All of these are listed at the end of this section.

During the final week leading up to the competition, we assigned ourselves crucial roles in order to improve our team workflow and accomplish as much as possible before we leave on Thursday, December 5. We've summarised a list of the things that we needed to get done during the last week:

In terms of the Robot:

1. More testing to make sure that intaking and scoring cubes works consistently
2. Final checks on intake
3. Passing inspection: no sharp edges or corners, fits within the size limit
4. Autonomous must fully complete and fully functional at least two days before leaving. This includes consistency of the match auton, as well as for programming skills. Whatever changes that needed to be made must be made as soon as possible. Also for skills auton, extremely important to get this done.

In terms of drivers and team members:

1. Practice practice practice. Both in terms of shooting around the field, pressure during a match, and driver skills.
2. Planning and using our time wisely. The schedule breakdown for this week:

In terms of packing:

1. Everything should be good to go by the end of Wednesday, Seba's primary job.
    a. N.B, this list is located at the end of the strategy section
2. Handling the robot inside the box, aka. bubble wrap etc.

In terms of the Design Notebook:

1. ALL PAGES from RoboRumble onwards must be completed by Tuesday night. The table of contents will also need to be updated for this reason.

2. Checklist must be FULLY UPDATED with mistakes and errors made from RoboRumble.
3. Everyone MUST have some coherence and practice for potential judge interviews. Since the competition is bigger, we're against more people. Design and Excellence are on the line.

In light of this, we also assigned ourselves crucial roles that we would have to stick to during the competition:

In terms of Roles during the competition:

1. Jason: Primary Driver. Is in charge of controlling the robot during matches and driver skills runs. He should be one of the people to de-stress the most, so he can focus on driving. He'll coordinate with the drivers of alliance partners to discuss any strategies and plans on the field itself. If we're not in a match, he's practicing or doing skills where permitted.
2. Justin: Builder and Quality Control. Main concern is to make sure the robot is intact after each match. Checks to see if any key parts of the robot have broken, and fixing them accordingly. Also in charge of alignment for the robot's autonomous programs, as any deviation from the path would mess up the auton routine.
3. Eion: Coach, Main scouter and Data Analysis. When we're at a match, coach us through what we have to do, and take note of robots that we may compete against in the future. When we're not competing, he would be scouting and talking with potential alliance members, some notable ones include the A teams of schools, or X teams. We need to get our name out there for other high level teams to know. Also in charge of going with Jason to talk to alliance partner before the match to work on strategy. Uses the given scouting information to decide on match decisions such as what blocks to pick up and other broader decisions such as viable alliance partners and team decisions.
4. John: Pit crew guy and Coder. Always keeping an eye on the parts on the table. Making sure there is always a charged set of batteries. Receiving data from Eion and updating the scouting sheet as needed. In charge of schedule and

suggesting when to do skills runs. Modifies the code as necessary if things don't work as planned.

5. Unfortunately, Seba will not be travelling with us due to personal and circumstantial factors. As such, he'll be contributing the most at home, through helping pack up materials and arranging our journal before we leave.

SPARE PARTS LIST FOR VEX FORMOSA 2019:

Note: things with asterisk (*) may have a larger number of parts for following the original creation of the parts list

VEX v5 Brains, Antennae, controllers

- 1 - spare brain
- 2 - spare controllers
- 2 - spare antennae

Screwdrivers, wrenches, and allen keys

- 3 - 5/64 screwdrivers
- 3 - 3/32 screwdrivers
- 4 - 5/64 allen keys
- 4 - 3/32 allen keys
- 5 - wrenches

Nuts and shaft collars

- 10 shaft collars*
- 20 keps nuts*

Screws

- 10 - 7mm screws*
- 20 - 12mm screws*
- 10 - 20mm screws*
- 6 - 25mm screws*

Metal parts

- Bars
  - 3 - 5 hole bars
  - 1 - 6 hole bar
  - 1 - 7 hole bar

- 1 - 8 hole bar
- 1 - 9 hole bar
- 1 - 10 hole bar

C channels

- 1 - 12 hole c channel
- 1 - 7 hole c channel
- 2 - 5 hole c channel

Standoffs

- 10 - small standoffs*
- 6 - 20mm standoffs*

Motors

3 - high speed motors

- 4 - smart cables*

Batteries

6 - v5 batteries*

Miscellaneous

10 - twist ties*

8 - blue rubber bands*

8 - thick rubber bands*

8 - thin rubber bands*

Scotch tape

Packaging tape

Scissors

8 white spacers (unknown length)*

APPENDIX

Program Code

```
/*----------------------------------------------------------------------------*/
/*                                                                            */
/*   Module:      main.cpp                                                    */
/*   Author:      abanesjo                                                    */
/*   Created:     Fri Aug 23 2019                                             */
/*   Description: V5 project                                                  */
/*                                                                            */
/*----------------------------------------------------------------------------*/
#include "vex.h"
#include "vex_global.h"
using namespace vex;

vex::controller controller1 = vex::controller();
vex::brain Brain;

vex::motor LeftFront = vex::motor(vex::PORT12);
vex::motor RightFront = vex::motor(vex::PORT19,true);
vex::motor LeftBack = vex::motor(vex::PORT13);
vex::motor RightBack = vex::motor(vex::PORT20,true);
vex::motor armLift = vex::motor(vex::PORT1);
vex::motor armLeft = vex::motor(vex::PORT4,true);
vex::motor armRight = vex::motor(vex::PORT6);
vex::motor ramp = vex::motor(vex::PORT2);

vex::vision VisionSensor = vex::vision(vex::PORT1);

#include "vex.h"
#include "vex_units.h"
#include <iostream>
#include <cmath>

using namespace vex;

competition Competition;

double armAngle = 0; //deprecated, used in old versions
int intakeState = 0;
```

```
int manual = 0; //deprecated, used in old versions
int countr = 0;
int y = 0; //deprecated, used in old versions
int rampState = 0;

void rampMode() {
 if(rampState == 0) {
   rampState = 1;
 } else if(rampState == 1) {
   rampState = 0;
 }
}
void intakeMode() {
 if(intakeState == 0) {
   intakeState = 1;
 } else if(intakeState == 1) {
   rampState = 0;
 }
}
#pragma region HelperTools
void velocityset(double left = 30, double right = 30) {
 LeftFront.setVelocity(left,vex::percentUnits::pct);
 LeftBack.setVelocity(left,vex::percentUnits::pct);
 RightFront.setVelocity(right,vex::percentUnits::pct);
 RightBack.setVelocity(right,vex::percentUnits::pct);
}
double todeg(double cm) {
 double wheelDiameter = 10.16;
 double circumference = wheelDiameter * 3.141592;
 double degreesToRotate = 360 * cm/circumference;
 return degreesToRotate;
}
double tocm(double deg) { //deprecated, used in old versions
 double wheelDiameter = 10.16;
 double circumference = wheelDiameter * 3.141592;
 double cm = deg * circumference /360;
 return cm;
}
void addAngle() { //deprecated, used in old versions
 armAngle +=1;
}
void subAngle() { //deprecated, used in old versions
 armAngle -= 1;
}
void printAngle(){ //used for troubleshooting
```

```
 Brain.Screen.clearScreen();
 Brain.Screen.printAt(100,150,"Angle: %0.2lf",ramp.rotation(vex::rotationUnits::deg));
}
void move(vex::directionType type) { //used in experimental versions
 LeftFront.spin(type);
 RightFront.spin(type);
 LeftBack.spin(type);
 RightBack.spin(type);
}
void forwardDistance(double cm, double vel) {
 velocityset(vel,vel);
 double deg = todeg(cm);
 LeftFront.rotateFor(deg, vex::rotationUnits::deg, false);
 LeftBack.rotateFor(deg, vex::rotationUnits::deg, false);
 RightFront.rotateFor(deg, vex::rotationUnits::deg, false);
 RightBack.rotateFor(deg, vex::rotationUnits::deg);
}
void forwardTime(double s, double vel) {
 velocityset(vel,vel);
 LeftFront.spin(vex::directionType::fwd);
 LeftBack.spin(vex::directionType::fwd);
 RightFront.spin(vex::directionType::fwd);
 RightBack.spin(vex::directionType::fwd);
 vex::task::sleep(s);
 LeftFront.stop();
 RightFront.stop();
 LeftBack.stop();
 RightBack.stop();
}
void forwardDistAccel(double dist, double initialVel, double finalVel) { //experimental, unfinished
 double deg = todeg(dist);
 double vel = initialVel;
 velocityset(vel, vel);
 double increment = (finalVel - initialVel)/100;
 double initial = LeftBack.rotation(vex::rotationUnits::deg);
 move(fwd);
 while(LeftBack.rotation(vex::rotationUnits::deg) < initial + dist) {
   double x1 = LeftBack.rotation(vex::rotationUnits::deg);
   vex::task::sleep(10);
   double x2 = LeftBack.rotation(vex::rotationUnits::deg);

   vel += increment;

   velocityset(vel, vel);
 }
```

```
 LeftFront.stop();
 RightFront.stop();
 LeftBack.stop();
 RightBack.stop();
}
//max velocity in m/s: 1.064
void backDistAccel(double dist, double timeToAccelerate, double initialVel, double finalVel) {
//experimental, unfinished
 double deg = todeg(dist) * -1;
 double vel = initialVel;
 velocityset(vel, vel);
 double increment = (finalVel - initialVel)/100;
 double status = LeftBack.rotation(vex::rotationUnits::deg);
 while(LeftBack.rotation(vex::rotationUnits::deg) > status + deg) {
   LeftFront.spin(vex::directionType::rev);
   RightFront.spin(vex::directionType::rev);
   LeftBack.spin(vex::directionType::rev);
   RightBack.spin(vex::directionType::rev);
   vex::task::sleep(timeToAccelerate/100);
   vel += increment;
   velocityset(vel, vel);
 }
 LeftFront.stop();
 RightFront.stop();
 LeftBack.stop();
 RightBack.stop();
}

void forwardAccel(double dist, double x1, double x2, double v1, double v2, double x3, double x4, double v3,
double v4) { //experimental, unfinished
 velocityset(v1, v1);
 double v = v1;
 double d1 = (v2 - v1)/100;
 double x0 = LeftBack.rotation(vex::rotationUnits::deg);
 double dx1 = (x2 - x1)/100;
 x1 += LeftBack.rotation(vex::rotationUnits::deg);
 x2 += LeftBack.rotation(vex::rotationUnits::deg);
 x3 += LeftBack.rotation(vex::rotationUnits::deg);
 x4 += LeftBack.rotation(vex::rotationUnits::deg);
 //acceleration
 move(fwd);
 while(LeftBack.rotation(vex::rotationUnits::deg) < x0 + dist) {
   while(x1 < LeftBack.rotation(vex::rotationUnits::deg) && LeftBack.rotation(vex::rotationUnits::deg) <
x2) {
     velocityset(v, v);
```

```
    vex::task::sleep(10);

  }
 }
 LeftFront.stop();
 RightFront.stop();
 LeftBack.stop();
 RightBack.stop();

}

void turn(bool directn, double degrees, double vel) {
 velocityset(vel,vel);
 if(directn == 0) { // right
   LeftFront.rotateFor(degrees, vex::rotationUnits::deg, false);
   LeftBack.rotateFor(degrees, vex::rotationUnits::deg, false);
   RightFront.rotateFor(-degrees, vex::rotationUnits::deg, false);
   RightBack.rotateFor(-degrees, vex::rotationUnits::deg);
 } else {//left
   LeftFront.rotateFor(-degrees, vex::rotationUnits::deg, false);
   LeftBack.rotateFor(-degrees, vex::rotationUnits::deg, false);
   RightFront.rotateFor(degrees, vex::rotationUnits::deg, false);
   RightBack.rotateFor(degrees, vex::rotationUnits::deg);
 }
}
void left() { //deprecated, used in old versions
 turn(0,300,30);
}
void right() { //deprecated, used in old versions
 turn(1,300,30);
}
void vSensor() { //experimental, may be discontinued
 //VisionSensor.takeSnapshot();

 if(VisionSensor.largestObject.exists) {
   Brain.Screen.print("Vision Sensor: X: %d",VisionSensor.largestObject.originX);
   Brain.Screen.print("Y: %d",VisionSensor.largestObject.originY);
   Brain.Screen.print("W: %d",VisionSensor.largestObject.width);
   Brain.Screen.print("H: %d",VisionSensor.largestObject.height);
 } else {
   Brain.Screen.print("Vision Sensor: Color Signature not Found!");
 }
 vex::task::sleep(200);
}
#pragma endregion
```

```
#pragma region actions
void setLeftExpo(vex::directionType type, int percentage) { //experimental, underdeveloped
 if(percentage >= 0) {
   percentage = 1.2 * pow(1.043,percentage) + 0.2*percentage - 1.2;
 } else {
   percentage = - percentage;
   percentage = 1.2 * pow(1.043,percentage) + 0.2*percentage - 1.2;
   percentage = -percentage;
 }
 LeftFront.spin(type, percentage, vex::velocityUnits::pct);
 LeftBack.spin(type, percentage, vex::velocityUnits::pct);
}
void setRightExpo(vex::directionType type, int percentage) { //experimental, underdeveloped
 if(percentage >= 0) {
   percentage = 1.2 * pow(1.043,percentage) + 0.2*percentage - 1.2;
 } else {
   percentage = - percentage;
   percentage = 1.2 * pow(1.043,percentage) + 0.2*percentage - 1.2;
   percentage = -percentage;
 }
 RightFront.spin(type, percentage, vex::velocityUnits::pct);
 RightBack.spin(type, percentage, vex::velocityUnits::pct);
}
void tank() {
 LeftFront.spin(vex::directionType::fwd, controller1.Axis3.position()*0.75, vex::percentUnits::pct);
 LeftBack.spin(vex::directionType::fwd, controller1.Axis3.position()*0.75, vex::percentUnits::pct);
 RightFront.spin(vex::directionType::fwd, controller1.Axis2.position()*0.75, vex::percentUnits::pct);
 RightBack.spin(vex::directionType::fwd, controller1.Axis2.position()*0.75, vex::percentUnits::pct);
}
void tankExpo() { //experimental, underdeveloped
 setLeftExpo(vex::directionType::fwd, (controller1.Axis3.value() + controller1.Axis4.value()));
 setRightExpo(vex::directionType::fwd, (controller1.Axis3.value() - controller1.Axis4.value()));
}
void arm() {
 ramp.setBrake(brakeType::hold);
 armLift.setBrake(brakeType::hold);
 ramp.setVelocity(45,vex::percentUnits::pct);
 armLift.setVelocity(100,vex::percentUnits::pct);

 if(rampState == 1) {//manual
   if(controller1.ButtonUp.pressing()) {
     ramp.spin(vex::directionType::fwd);
   } else if(controller1.ButtonDown.pressing()) {
     ramp.spin(vex::directionType::rev);
```

```
  } else {
    if(armLift.rotation(vex::rotationUnits::deg) > 50) {
      countr = 1;
      ramp.rotateTo(350,vex::rotationUnits::deg,false);
    } else if(armLift.rotation(vex::rotationUnits::deg) < 50 && countr == 1) {
      ramp.rotateTo(10,vex::rotationUnits::deg);
      countr = 0;
    } else {
      ramp.stop();
    }
    ramp.stop();
  }

    if(controller1.ButtonR1.pressing()) {
    armLift.spin(vex::directionType::fwd);
  } else if(controller1.ButtonR2.pressing()) {
    armLift.spin(vex::directionType::rev);
  } else {
    armLift.stop();
  }
 }
//-------------------------
 else {//automatic
  if(controller1.ButtonUp.pressing()) {
    ramp.rotateTo(1300,vex::rotationUnits::deg,false);
  } else if(controller1.ButtonDown.pressing()) {
    ramp.rotateTo(10,vex::rotationUnits::deg,false);
  } else {
    if(armLift.rotation(vex::rotationUnits::deg) > 30) {
      countr = 1;
      ramp.rotateTo(500,vex::rotationUnits::deg,false);
    } else if(armLift.rotation(vex::rotationUnits::deg) < 30 && countr == 1) {
      ramp.rotateTo(10,vex::rotationUnits::deg,false);
    }
  }

 if(controller1.ButtonX.pressing()) {
  armLift.rotateTo(1000,vex::rotationUnits::deg,false);
 } else if(controller1.ButtonA.pressing()) {
  armLift.rotateTo(760,vex::rotationUnits::deg,false);
 } else if(controller1.ButtonB.pressing()) {
  armLift.rotateTo(10,vex::rotationUnits::deg,false);
 }

 }
```

```
 //---------------------

 printAngle(); //for troubleshooting
 if(controller1.ButtonLeft.pressing()) {
   intakeState = 0;
 } else if(controller1.ButtonRight.pressing()) {
   intakeState = 1;
 }

 if(controller1.ButtonL1.pressing()) {
   armLeft.setVelocity(100,vex::percentUnits::pct);
   armRight.setVelocity(100,vex::percentUnits::pct);
   armLeft.spin(vex::directionType::fwd);
   armRight.spin(vex::directionType::fwd);
 } else if(controller1.ButtonL2.pressing()) {
   armLeft.setVelocity(100,vex::percentUnits::pct);
   armRight.setVelocity(100,vex::percentUnits::pct);
   armLeft.spin(vex::directionType::rev);
   armRight.spin(vex::directionType::rev);
 } else {
   if(intakeState == 0) {
     armLeft.stop();
     armRight.stop();
   } else {
     armLeft.spin(vex::directionType::fwd,100,vex::velocityUnits::pct);
     armRight.spin(vex::directionType::fwd,100,vex::velocityUnits::pct);
   }
 }
}
#pragma endregion


#pragma region auton
void skill() { //10 points, old
 //autonomous skills code. Code for the actual competition is yet to be
 //implemented as here the robot doesn't start from a base.
 //expansion

 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::rev);
 armRight.spin(vex::directionType::rev);
 vex::task::sleep(500);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
```

```
forwardTime(300,-40);
vex::task::sleep(150);

//end of expansion
armLeft.setVelocity(100,vex::percentUnits::pct);
armRight.setVelocity(100,vex::percentUnits::pct);
armLeft.spin(vex::directionType::fwd);//
armRight.spin(vex::directionType::fwd);
forwardDistance(80,10);
armLeft.stop();
armRight.stop();
vex::task::sleep(500);

armLeft.setVelocity(5,vex::percentUnits::pct);
armRight.setVelocity(5,vex::percentUnits::pct);
armLeft.spin(vex::directionType::fwd);
armRight.spin(vex::directionType::fwd);
forwardDistance(-20,30);
vex::task::sleep(500);
turn(1,405,30); //389
vex::task::sleep(500);
// armLeft.setVelocity(-20,vex::percentUnits::pct);
// armRight.setVelocity(-20,vex::percentUnits::pct);
// armLeft.spin(vex::directionType::fwd);
// armRight.spin(vex::directionType::fwd);
// vex::task::sleep(1000);

forwardTime(3500,50);
vex::task::sleep(500);

forwardDistance(-4,15);

armLeft.setVelocity(25,vex::percentUnits::pct);
armRight.setVelocity(25,vex::percentUnits::pct);
armLeft.rotateFor(-200,vex::rotationUnits::deg,false);
armRight.rotateFor(-200,vex::rotationUnits::deg);

armLeft.stop();
armRight.stop();

vex::task::sleep(500);

// armLeft.setVelocity(-5,vex::percentUnits::pct);
// armRight.setVelocity(-5,vex::percentUnits::pct);
// armLeft.spin(vex::directionType::fwd);
```

```
// armRight.spin(vex::directionType::fwd);
ramp.setVelocity(45,vex::percentUnits::pct);
ramp.rotateFor(1100,vex::rotationUnits::deg);
vex::task::sleep(500);
forwardTime(1000,15);
vex::task::sleep(500);
armLeft.setVelocity(-60,vex::percentUnits::pct);
armRight.setVelocity(-60,vex::percentUnits::pct);
armLeft.spin(vex::directionType::fwd);
armRight.spin(vex::directionType::fwd);
forwardDistance(-45,60); // 25
armLeft.stop();
armRight.stop();
vex::task::sleep(1000);
//-------------------------------------
turn(1,410,15); //maybe 405
LeftFront.setVelocity(30,vex::percentUnits::pct);
LeftBack.setVelocity(30,vex::percentUnits::pct);
RightFront.setVelocity(30,vex::percentUnits::pct);
RightBack.setVelocity(30,vex::percentUnits::pct);
vex::task::sleep(500);
ramp.rotateFor(-1000,vex::rotationUnits::deg,false);
forwardTime(2500,-30);
vex::task::sleep(500);
//---------------------------------------
forwardDistance(102,10);
vex::task::sleep(500);
armLeft.setVelocity(10,vex::percentUnits::pct);
armRight.setVelocity(10,vex::percentUnits::pct);
armLeft.rotateFor(300,vex::rotationUnits::deg,false);
armRight.rotateFor(300,vex::rotationUnits::deg);

vex::task::sleep(500);
forwardDistance(-10,10);
vex::task::sleep(500);

armLift.setVelocity(30,vex::percentUnits::pct);
armLift.rotateTo(1200,vex::rotationUnits::deg);
vex::task::sleep(1000);
//forwardDistance(45,30);
forwardTime(3000,30);
vex::task::sleep(500);
armLift.rotateTo(700,vex::rotationUnits::deg);
vex::task::sleep(500);
armLeft.setVelocity(-25,vex::percentUnits::pct);
```

```
 armRight.setVelocity(-25,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
 forwardDistance(-80,30);
 armLift.stop();
 armLeft.stop();
 armRight.stop();
}

void skillsixteen() { //newer version, 16 points
 //---------Expansion---------------
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::rev);
 armRight.spin(vex::directionType::rev);
 vex::task::sleep(500);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
 forwardTime(300,-40);
 vex::task::sleep(150);
 //----------------Intake----------------
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);//
 armRight.spin(vex::directionType::fwd);
 forwardDistance(75,15);
 vex::task::sleep(500);
 forwardDistance(-10,20);
 vex::task::sleep(500);
 turn(0,147,20);
 vex::task::sleep(500);
 forwardDistance(-72,20);
 vex::task::sleep(500);
 turn(1,147,20);
 vex::task::sleep(500);
 //----------------
 forwardDistance(78,15);
 armLeft.setBrake(hold);
 armRight.setBrake(hold);
 armLeft.stop();
 armRight.stop();
 vex::task::sleep(500);
 forwardDistance(-38,15);

 turn(1,380,20); //389
```

```
vex::task::sleep(500);
// armLeft.setVelocity(-20,vex::percentUnits::pct);
// armRight.setVelocity(-20,vex::percentUnits::pct);
// armLeft.spin(vex::directionType::fwd);
// armRight.spin(vex::directionType::fwd);
// vex::task::sleep(1000);

forwardTime(2500,50);
vex::task::sleep(500);

forwardTime(500,-10);

armLeft.setVelocity(25,vex::percentUnits::pct);
armRight.setVelocity(25,vex::percentUnits::pct);
armLeft.rotateFor(-75,vex::rotationUnits::deg,false);
armRight.rotateFor(-75,vex::rotationUnits::deg);

armLeft.stop();
armRight.stop();

vex::task::sleep(500);

armLeft.setVelocity(20,vex::percentUnits::pct);
armRight.setVelocity(20,vex::percentUnits::pct);
armLeft.spin(vex::directionType::fwd);
armRight.spin(vex::directionType::fwd);
ramp.setVelocity(30,vex::percentUnits::pct);
ramp.rotateTo(1300,vex::rotationUnits::deg);
vex::task::sleep(500);
forwardTime(800,10);
vex::task::sleep(500);
armLeft.setVelocity(-15,vex::percentUnits::pct);
armRight.setVelocity(-15,vex::percentUnits::pct);
// armLeft.spin(vex::directionType::fwd);
// armRight.spin(vex::directionType::fwd);
armLeft.rotateFor(-800,vex::rotationUnits::deg,false);
armRight.rotateFor(-800,vex::rotationUnits::deg,false);

forwardDistance(-30,30); // 25
armLeft.stop();
armRight.stop();

//--------8 points-------------
 vex::task::sleep(500);
turn(1,430,15); //maybe 405
```

```
LeftFront.setVelocity(30,vex::percentUnits::pct);
LeftBack.setVelocity(30,vex::percentUnits::pct);
RightFront.setVelocity(30,vex::percentUnits::pct);
RightBack.setVelocity(30,vex::percentUnits::pct);
vex::task::sleep(500);
ramp.rotateTo(400,vex::rotationUnits::deg,false);
forwardTime(2500,-50);
vex::task::sleep(500);
//---------------------------------------
forwardDistance(102,20);
vex::task::sleep(500);
armLeft.setVelocity(40,vex::percentUnits::pct);
armRight.setVelocity(40,vex::percentUnits::pct);
armLeft.rotateFor(500,vex::rotationUnits::deg,false);
armRight.rotateFor(500,vex::rotationUnits::deg);

vex::task::sleep(500);
forwardDistance(-10,20);
vex::task::sleep(500);
 //Old method of "slam-dunking" the cube into the tower
// armLift.setVelocity(50,vex::percentUnits::pct);
// armLift.rotateTo(1200,vex::rotationUnits::deg);
// vex::task::sleep(200);
// forwardTime(1700,30);
// vex::task::sleep(300);
// armLift.rotateTo(700,vex::rotationUnits::deg);
// vex::task::sleep(300);
// armLeft.setVelocity(-25,vex::percentUnits::pct);
// armRight.setVelocity(-25,vex::percentUnits::pct);
// armLeft.spin(vex::directionType::fwd);
// armRight.spin(vex::directionType::fwd);

armLift.setVelocity(50,vex::percentUnits::pct);
armLift.rotateTo(900,vex::rotationUnits::deg);
vex::task::sleep(200);
forwardDistance(30,20);
vex::task::sleep(500);
armLeft.setVelocity(-100,vex::percentUnits::pct);
armRight.setVelocity(-100,vex::percentUnits::pct);
armLeft.spin(vex::directionType::fwd);
armRight.spin(vex::directionType::fwd);
vex::task::sleep(1000);
forwardDistance(-80,30);
armLift.stop();
armLeft.stop();
```

```
 armRight.stop();


}
void bluEightSlow() {
 //---------------
 //expansion
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::rev);
 armRight.spin(vex::directionType::rev);
 vex::task::sleep(500);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
 forwardTime(300,-40);
 vex::task::sleep(150);

 //end of expansion
 //----------------
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);//
 armRight.spin(vex::directionType::fwd);
 forwardDistance(75,15);
 vex::task::sleep(500);
 forwardDistance(-10,20);
 vex::task::sleep(500);
 turn(0,135,20);
 vex::task::sleep(500);
 forwardDistance(-78,20);
 vex::task::sleep(500);
 turn(1,135,20);
 vex::task::sleep(500);
 //----------------
 forwardDistance(78,15);
 armLeft.setBrake(hold);
 armRight.setBrake(hold);
 armLeft.stop();
 armRight.stop();
 vex::task::sleep(500);
 forwardDistance(-38,15);

 turn(1,380,20); //389
 vex::task::sleep(500);
 // armLeft.setVelocity(-20,vex::percentUnits::pct);
```

```
 // armRight.setVelocity(-20,vex::percentUnits::pct);
 // armLeft.spin(vex::directionType::fwd);
 // armRight.spin(vex::directionType::fwd);
 // vex::task::sleep(1000);

 forwardTime(2500,50);
 vex::task::sleep(500);

 forwardTime(500,-10);

 armLeft.setVelocity(25,vex::percentUnits::pct);
 armRight.setVelocity(25,vex::percentUnits::pct);
 armLeft.rotateFor(-50,vex::rotationUnits::deg,false);
 armRight.rotateFor(-50,vex::rotationUnits::deg);

 armLeft.stop();
 armRight.stop();

 vex::task::sleep(500);

 armLeft.setVelocity(20,vex::percentUnits::pct);
 armRight.setVelocity(20,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
 ramp.setVelocity(30,vex::percentUnits::pct);
 ramp.rotateTo(1300,vex::rotationUnits::deg);
 vex::task::sleep(500);
 forwardTime(800,10);
 vex::task::sleep(500);
 armLeft.setVelocity(-15,vex::percentUnits::pct);
 armRight.setVelocity(-15,vex::percentUnits::pct);
 // armLeft.spin(vex::directionType::fwd);
 // armRight.spin(vex::directionType::fwd);
 armLeft.rotateFor(-800,vex::rotationUnits::deg,false);
 armRight.rotateFor(-800,vex::rotationUnits::deg,false);
 forwardDistance(-45,60); // 25
 armLeft.stop();
 armRight.stop();
}
void go() {
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);//
 armRight.spin(vex::directionType::fwd);
 forwardDistance(120,30);
```

```
 }
void bluEightOld() {
  //---------------
 //expansion
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::rev);
 armRight.spin(vex::directionType::rev);
 vex::task::sleep(500);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
 forwardTime(300,-40);
 vex::task::sleep(150);

 //end of expansion
 //----------------
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);//
 armRight.spin(vex::directionType::fwd);
 //intake
 forwardDistance(85,30);
 vex::task::sleep(100);
 forwardDistance(-30,40);
 vex::task::sleep(200);
 turn(0,180,20);
 vex::task::sleep(150);
 forwardDistance(-74,40);
 vex::task::sleep(250);
 turn(1,175,20);
 vex::task::sleep(150);

 forwardDistance(90,30);
 armLeft.setBrake(hold);
 armRight.setBrake(hold);

 armLeft.setVelocity(15,vex::percentUnits::pct);
 armRight.setVelocity(15,vex::percentUnits::pct);
 forwardDistance(-50,40);
 vex::task::sleep(400);
 //--------------------Stacking--------------
 turn(1,400,20); //389
 vex::task::sleep(150);
 armLeft.stop();
 armRight.stop();
```

```
 ramp.setVelocity(60,vex::percentUnits::pct);
 ramp.rotateTo(1300,vex::rotationUnits::deg,false);
 forwardTime(2500,60);
 vex::task::sleep(500);
 armLeft.setVelocity(-15,vex::percentUnits::pct);
 armRight.setVelocity(-15,vex::percentUnits::pct);
 armLeft.rotateFor(-800,vex::rotationUnits::deg,false);
 armLeft.rotateFor(-800,vex::rotationUnits::deg,false);
 forwardDistance(-30,40); // 25
 armLeft.stop();
 armRight.stop();
}
void bluEight() {
 //---------------Expansion
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::rev);
 armRight.spin(vex::directionType::rev);
 vex::task::sleep(500);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
 forwardTime(300,-40);
 vex::task::sleep(150);
 //----------------intake---------
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);//
 armRight.spin(vex::directionType::fwd);

 forwardDistance(85,35);
 vex::task::sleep(200);
 turn(1,130,25);
 vex::task::sleep(200);
 forwardDistance(-80,40);
 vex::task::sleep(250);
 turn(0,130,25);
 vex::task::sleep(250);
 forwardDistance(70,30);

 armLeft.setBrake(hold);
 armRight.setBrake(hold);

 armLeft.setVelocity(15,vex::percentUnits::pct);
 armRight.setVelocity(15,vex::percentUnits::pct);
```

```
 turn(1,390,30);
 vex::task::sleep(300);
 //---------------------------
 ramp.setVelocity(25,vex::percentUnits::pct);
 ramp.rotateTo(1300,vex::rotationUnits::deg,false);
 forwardTime(2500,50);
 forwardTime(400,-10);
 vex::task::sleep(100);
 ramp.setVelocity(50,vex::percentUnits::pct);
 armLeft.setVelocity(25,vex::percentUnits::pct);
 armRight.setVelocity(25,vex::percentUnits::pct);
 armLeft.rotateFor(-70,vex::rotationUnits::deg,false);
 armRight.rotateFor(-70,vex::rotationUnits::deg);
 vex::task::sleep(100);
 forwardTime(400,10);
 vex::task::sleep(100);
 armLeft.setVelocity(-15,vex::percentUnits::pct);
 armRight.setVelocity(-15,vex::percentUnits::pct);
 armLeft.rotateFor(-800,vex::rotationUnits::deg,false);
 armLeft.rotateFor(-800,vex::rotationUnits::deg,false);
 forwardDistance(-30,40); // 25
 armLeft.stop();
 armRight.stop();
}
void bluSafeSlow() {
   //autonomous skills code. Code for the actual competition is yet to be
 //implemented as here the robot doesn't start from a base.
  //expansion
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::rev);
 armRight.spin(vex::directionType::rev);
 vex::task::sleep(500);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
 forwardTime(300,-40);
 vex::task::sleep(150);

 //end of expansion
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);//
 armRight.spin(vex::directionType::fwd);
 forwardDistance(83,20);
```

```
armLeft.stop();
armRight.stop();
vex::task::sleep(500);

armLeft.setVelocity(5,vex::percentUnits::pct);
armRight.setVelocity(5,vex::percentUnits::pct);
armLeft.spin(vex::directionType::fwd);
armRight.spin(vex::directionType::fwd);
forwardDistance(-23,60);
vex::task::sleep(500);
turn(1,380,20); //389
vex::task::sleep(500);
// armLeft.setVelocity(-20,vex::percentUnits::pct);
// armRight.setVelocity(-20,vex::percentUnits::pct);
// armLeft.spin(vex::directionType::fwd);
// armRight.spin(vex::directionType::fwd);
// vex::task::sleep(1000);

forwardTime(1500,70);
vex::task::sleep(200);
forwardTime(550,-10);

armLeft.setVelocity(25,vex::percentUnits::pct);
armRight.setVelocity(25,vex::percentUnits::pct);
armLeft.rotateFor(-450,vex::rotationUnits::deg,false); // 130
armRight.rotateFor(-450,vex::rotationUnits::deg); // 130

vex::task::sleep(500);

armLeft.setVelocity(15,vex::percentUnits::pct);
armRight.setVelocity(15,vex::percentUnits::pct);
ramp.setVelocity(60,vex::percentUnits::pct);
ramp.rotateTo(750,vex::rotationUnits::deg);
vex::task::sleep(500);
armLeft.rotateFor(300,vex::rotationUnits::deg,false); // 130
armRight.rotateFor(300,vex::rotationUnits::deg);
vex::task::sleep(500);
ramp.rotateTo(1300,vex::rotationUnits::deg);
vex::task::sleep(500);
forwardTime(800,15);
vex::task::sleep(800);
// armLeft.setVelocity(-15,vex::percentUnits::pct);
// armRight.setVelocity(-15,vex::percentUnits::pct);
// armLeft.spin(vex::directionType::fwd);
// armRight.spin(vex::directionType::fwd);
```

```
 // armLeft.rotateFor(-800,vex::rotationUnits::deg,false);
 // armRight.rotateFor(-800,vex::rotationUnits::deg,false);
 forwardDistance(-45,30); // 25
 armLeft.stop();
 armRight.stop();
 vex::task::sleep(1000);
}
void bluSafe() { //5 points
 //autonomous skills code. Code for the actual competition is yet to be
 //implemented as here the robot doesn't start from a base.
  //expansion
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::rev);
 armRight.spin(vex::directionType::rev);
 vex::task::sleep(500);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
 forwardTime(300,-40);
 vex::task::sleep(150);

 //end of expansion
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);//
 armRight.spin(vex::directionType::fwd);
 forwardDistance(85,22);
 armLeft.stop();
 armRight.stop();
 vex::task::sleep(300);

 armLeft.setVelocity(15,vex::percentUnits::pct);
 armRight.setVelocity(15,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
 forwardDistance(-25,60);
 vex::task::sleep(500);
 turn(1,380,20); //389
 vex::task::sleep(500);
 // armLeft.setVelocity(-20,vex::percentUnits::pct);
 // armRight.setVelocity(-20,vex::percentUnits::pct);
 // armLeft.spin(vex::directionType::fwd);
 // armRight.spin(vex::directionType::fwd);
 // vex::task::sleep(1000);
```

```
 forwardTime(1500,70);
 vex::task::sleep(200);
 forwardTime(550,-10);

 armLeft.setVelocity(15,vex::percentUnits::pct);
 armRight.setVelocity(15,vex::percentUnits::pct);
 armLeft.rotateFor(-450,vex::rotationUnits::deg,false); // 130
 armRight.rotateFor(-450,vex::rotationUnits::deg,false); // 130

 ramp.setVelocity(40,vex::percentUnits::pct);
 ramp.rotateTo(1300,vex::rotationUnits::deg);
 vex::task::sleep(500);
 forwardTime(800,15);
 vex::task::sleep(500);
 // armLeft.setVelocity(-15,vex::percentUnits::pct);
 // armRight.setVelocity(-15,vex::percentUnits::pct);
 // armLeft.spin(vex::directionType::fwd);
 // armRight.spin(vex::directionType::fwd);
 // armLeft.rotateFor(-800,vex::rotationUnits::deg,false);
 // armRight.rotateFor(-800,vex::rotationUnits::deg,false);
 forwardDistance(-45,30); // 25
 armLeft.stop();
 armRight.stop();
}
void reedSafe() {//5pts, red
 //autonomous skills code. Code for the actual competition is yet to be
 //implemented as here the robot doesn't start from a base.
  //expansion
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::rev);
 armRight.spin(vex::directionType::rev);
 vex::task::sleep(500);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
 forwardTime(300,-40);
 vex::task::sleep(150);

 //end of expansion
 armLeft.setVelocity(100,vex::percentUnits::pct);
 armRight.setVelocity(100,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);//
 armRight.spin(vex::directionType::fwd);
 forwardDistance(85,22);
 armLeft.stop();
```

```
 armRight.stop();
 vex::task::sleep(300);

 armLeft.setVelocity(15,vex::percentUnits::pct);
 armRight.setVelocity(15,vex::percentUnits::pct);
 armLeft.spin(vex::directionType::fwd);
 armRight.spin(vex::directionType::fwd);
 forwardDistance(-25,60);
 vex::task::sleep(500);
 turn(0,380,20); //389
 vex::task::sleep(500);
 // armLeft.setVelocity(-20,vex::percentUnits::pct);
 // armRight.setVelocity(-20,vex::percentUnits::pct);
 // armLeft.spin(vex::directionType::fwd);
 // armRight.spin(vex::directionType::fwd);
 // vex::task::sleep(1000);

 forwardTime(1500,70);
 vex::task::sleep(200);
 forwardTime(550,-10);

 armLeft.setVelocity(15,vex::percentUnits::pct);
 armRight.setVelocity(15,vex::percentUnits::pct);
 armLeft.rotateFor(-450,vex::rotationUnits::deg,false); // 130
 armRight.rotateFor(-450,vex::rotationUnits::deg,false); // 130

 ramp.setVelocity(25,vex::percentUnits::pct);
 ramp.rotateTo(1300,vex::rotationUnits::deg);
 vex::task::sleep(500);
 forwardTime(800,15);
 vex::task::sleep(500);
 // armLeft.setVelocity(-15,vex::percentUnits::pct);
 // armRight.setVelocity(-15,vex::percentUnits::pct);
 // armLeft.spin(vex::directionType::fwd);
 // armRight.spin(vex::directionType::fwd);
 // armLeft.rotateFor(-800,vex::rotationUnits::deg,false);
 // armRight.rotateFor(-800,vex::rotationUnits::deg,false);
 forwardDistance(-45,30); // 25
 armLeft.stop();
 armRight.stop();
}
#pragma endregion
#pragma region compTemplate

void pre_auton(void) {
```

```
 armLift.resetRotation();
 ramp.resetRotation();
}
void usercontrol() {
 controller1.ButtonY.pressed(rampMode);
 controller1.ButtonRight.pressed(intakeMode);
 while(true) {
   arm();
   tank();
   vex::task::sleep(10);
 }
}
void autonomous(void) {
 bluEight();
 //skillsixteen();
}

int main() {
 Competition.autonomous(autonomous);
 Competition.drivercontrol(usercontrol);

 pre_auton();

 while(1) {
   vex::task::sleep(100);
 }

}
#pragma endregion
```