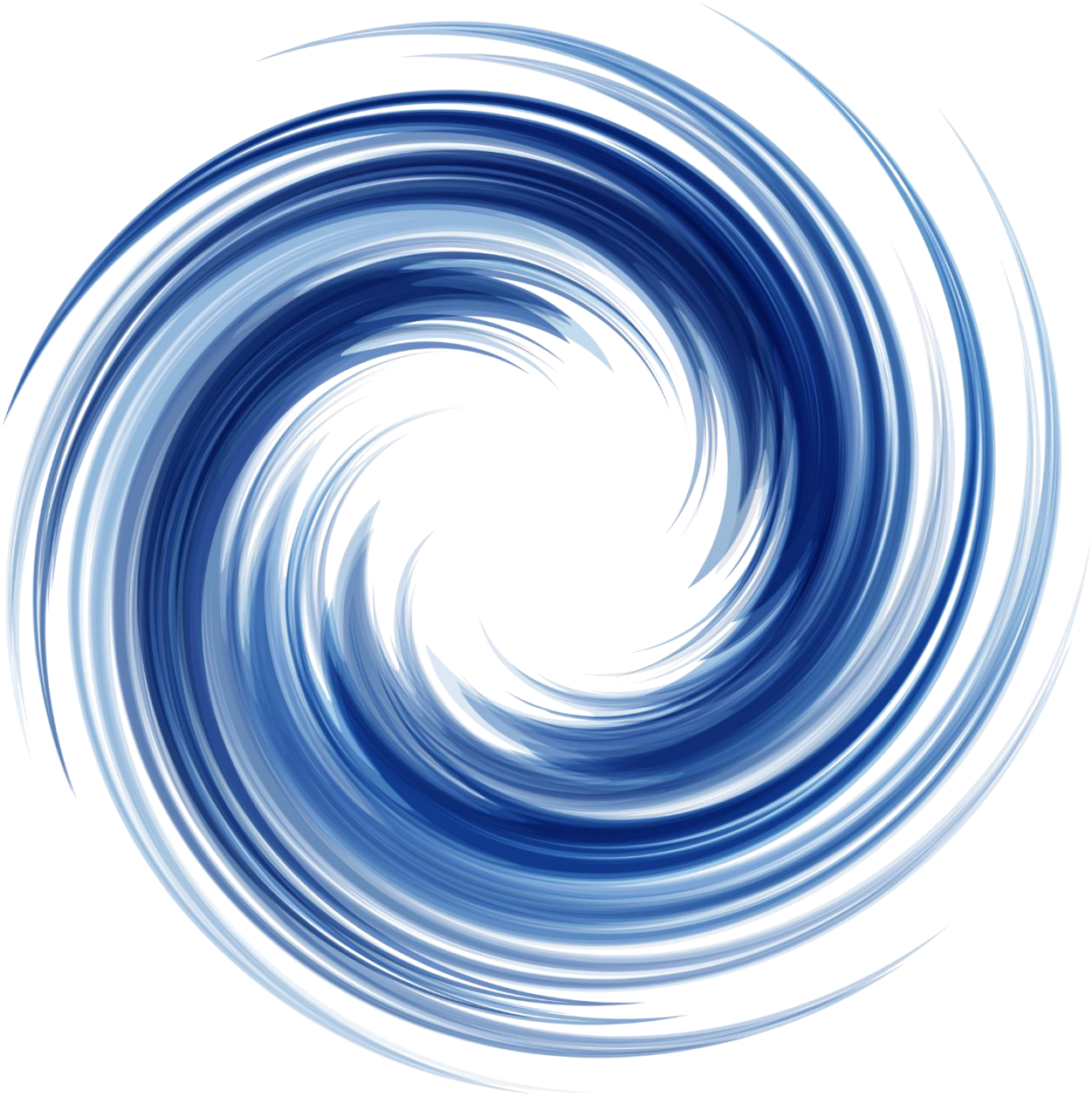
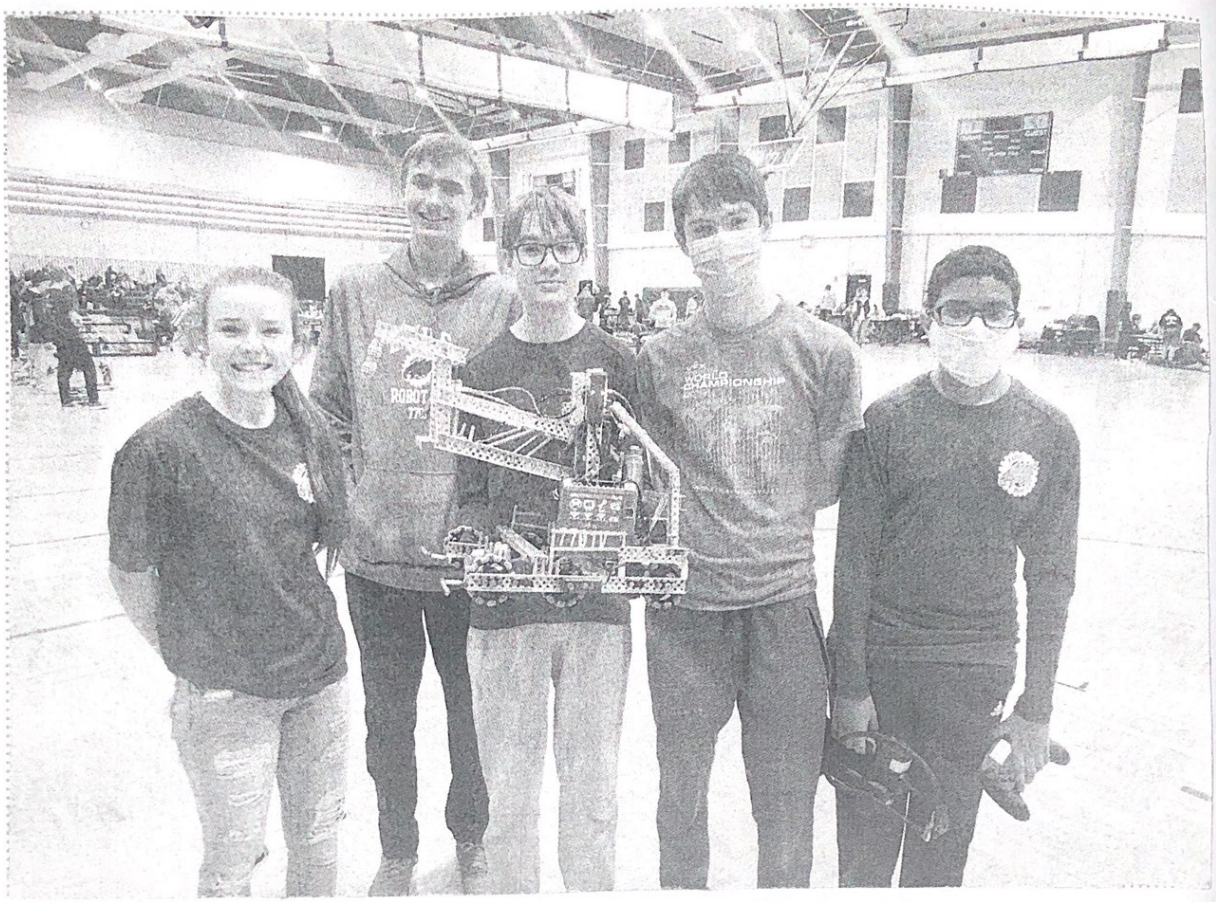


7701V | Vortex

2021-2022 Tipping Point

Engineering Notebook

Team Photo



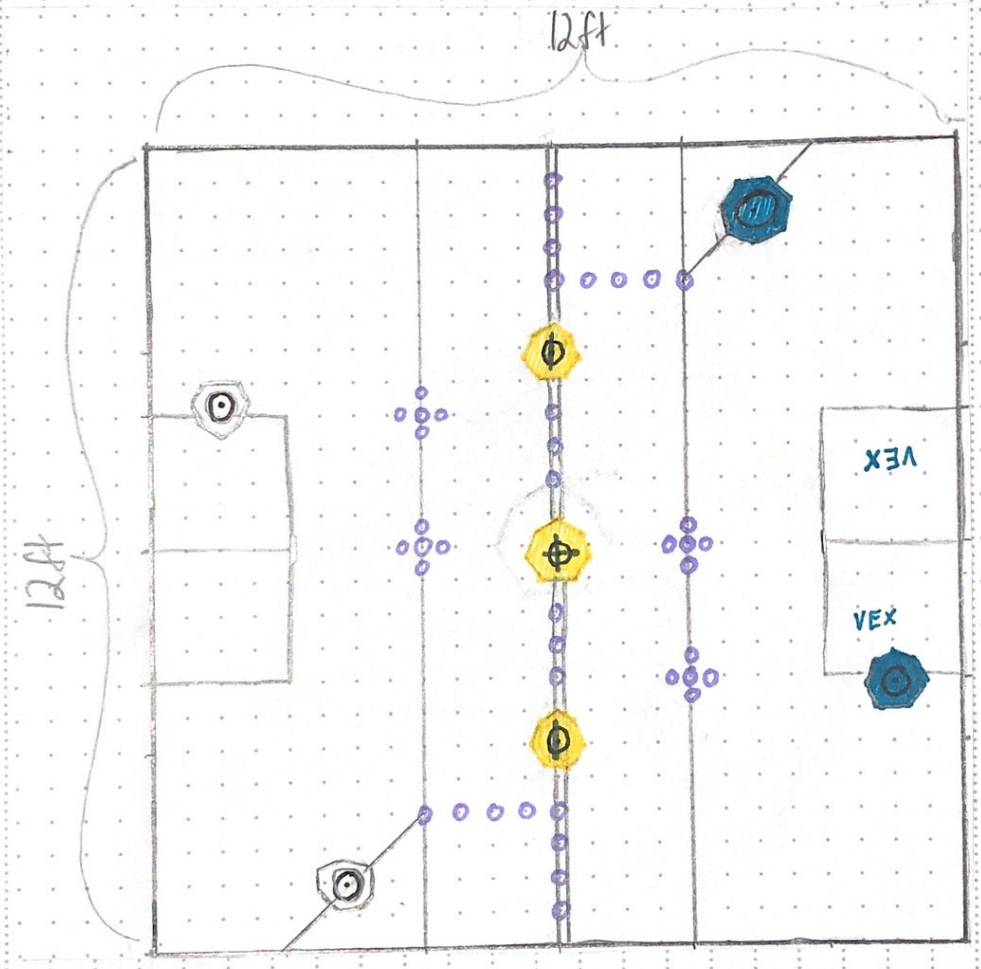
Team Profile

- Drew Kachur
  - Driver
  - Junior
  - 4th year of Robotics
- Jacob Walter
  - Builder
  - Senior
  - 3rd year of robotics
- Mayank Patibandla
  - Programmer
  - Junior
  - 5th year of Robotics
- Olivia Mulholland
  - Documentation Specialist
  - Junior
  - 4th year of robotics
- Thomas Wallbank
  - Scout
  - Junior
  - 4th year of Robotics

My Projects

page	project	date
1	Game Description	8/17/21
2	Scoring	8/17
3	Potential Strategies	8/17
5	Team Goals	8/17
7	Chassis Design Plan	8/17
8	Base & wheels	8/19-24
9	Ring Lift & Bar Bar	8/31-9/7
10	9/14-23 Daily logs	9/14-23
13	Kills of all kinds (Update on p. 19)	9/30-10/19
18	Early Practice Strategy	10/28
20	First comp Prep., Thoughts and Adjustments	11/2-11/16
25	11/20 Tournament Prep & Reflection	11/18-11/23
27	Rebuild plan	11/25-1/4
31	Comps Breakdown & Changes	1/11-1/24
36	Daily logs 1/25-27	1/25-27
39	Update & Driving Strategy	1/28
40	1/29 Comp. Break Down & Alliance Choosing Strategy	1/29

Vex 2021-22 Game: Tipping Point



Objective: Attain a higher score than opposing alliance by scoring Rings, moving Mobil Goals to Alliance Zones, and by elevating on Platforms at the end of the match.

project designed by: witnessed by:

date: 8/7/21

Scoring

- Each ring scored on a Neutral Mobile Goal High Branch - 10 points
- Each ring scored on any other Mobile Goal Branch - 3 points
- Each ring scored in a Mobile Goal base - 1 point
- Each Mobile Goal scored in an Alliance Home Zone - 20 points
- Each robot that is elevated - 30 points
- Each Mobile Goal that is elevated - 40 points
- One Ring scored on/in each alliance Mobile goal and a cleared AWP Line in Autonomous - 1 Win Point

There will be a 15 second autonomous period at the beginning of each match. The Alliance that scores the most gets 20 points added to their final score.

project

designed by:

witnessed by:

date: 8/17/21

Potential Strategies

Autonomus (15 second)

- go for the large Mobile goal first
- get the win point

Alliance

- one team focused on rings, while other works on putting the mobile goals on the platform

project designed by: witnessed by:

date: 8/17/21

Team Goals

- Win divison finals at Worlds
- Win design award
- Make it to the Final Match of a tournament ✓ achieved 1/8
-

project designed by: witnessed by:

date: 8/17/21

Chassis Design Plan

Musts

- 30x30 → not too big, but still big enough
- Gear system on wheels for speed
- Place for sweep intake
- Clearance for Goal lift

} Opposite sides

Omni Wheels vs Smooth Wheels

Omni Wheels		Smooth Wheels	
Pros	Cons	Pros	Cons
-easy turning	-less control -less traction -easily pushed by bots	-Traction -Control	-turns not as smooth -less maneuverable

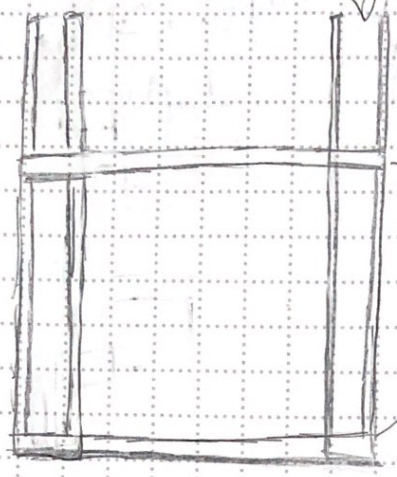
8/26

After using the smooth wheels + having a difficulty turning, made the decision to use Omni Wheels.

project designed by: witnessed by:

date: 8/17

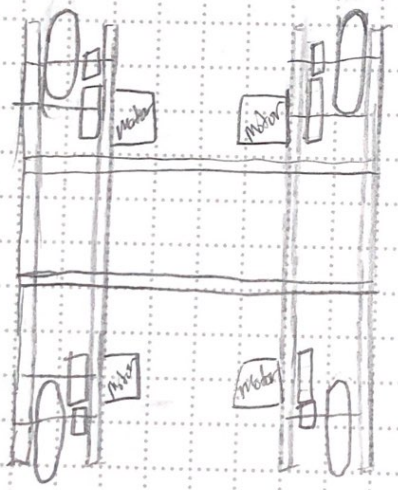
Base 8/19



area for wheels

support bars

Chassis w/wheels 8/24



- Support bars moved for wheels
- Empty space left for a ring intake

project

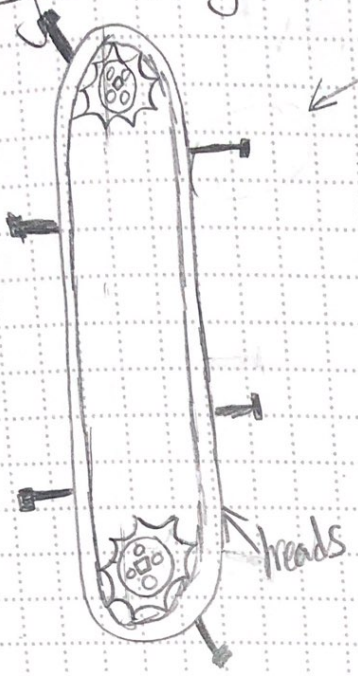
designed by:

witnessed by:

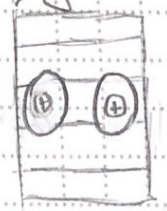
date: 8/19-24

Ring lift design 8/31

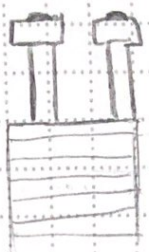
Side view



Sky view

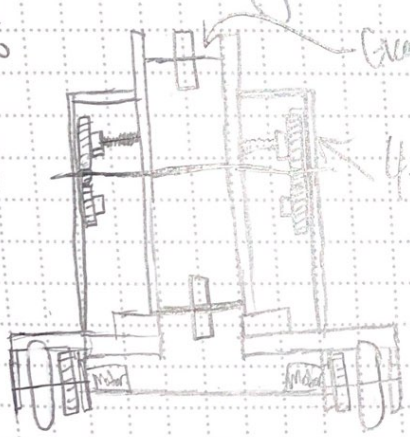


Front view



Far bar + Ring lift 9/7

Back view



project designed by: witnessed by:

date: 8/31-9/7

9/14 log

- completed ring lift (separate from robot)
- had to change spacer size on ring lift for guard clearance
- failed at putting ring lift on robot
- Mayank worked on Odom

Moving Forward

- redesign mounting system for ring lift
- Begin prototyping front intake

project

designed by:

witnessed by:

date: 9/14

9/16 log

- reattached main lift → discovered that a side approach was more effective & provided more space for scoring to cycle

Moving Forward

- Prototype front intake
- Smooth ring intake

Possible problems to Watch for

- inconsistency in ring lift
- in spec?
- collision between front intake & Main lift

project designed by: witnessed by:

date: 9/16

9/23

- Completely eliminated front intake
- Mounted stabilization bar to ring fit

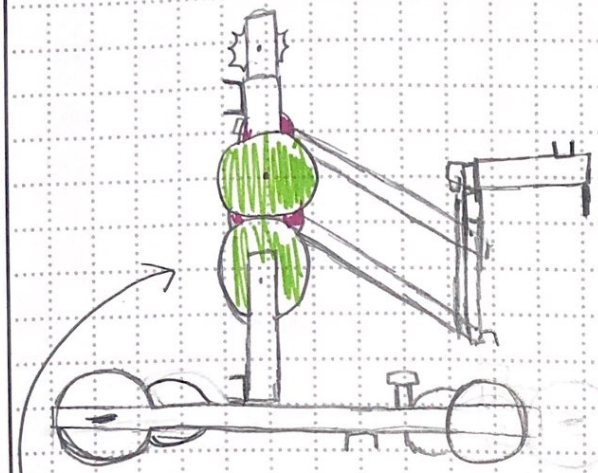
project

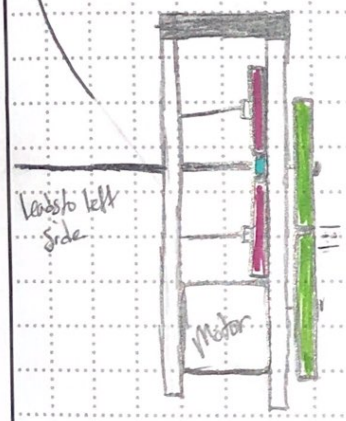
designed by:

witnessed by:

date: 9/23

4-Bar Lift + Gear System





Motor leads to 84 toothed gear, which leads to another 84 toothed gear, on the same axel as 12 tooth, 12 then leads to a 60 above it below it.

1:1:1:5

project

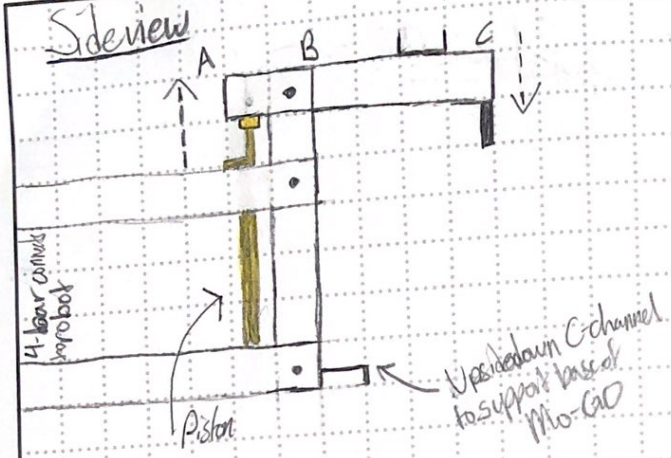
designed by:

witnessed by:

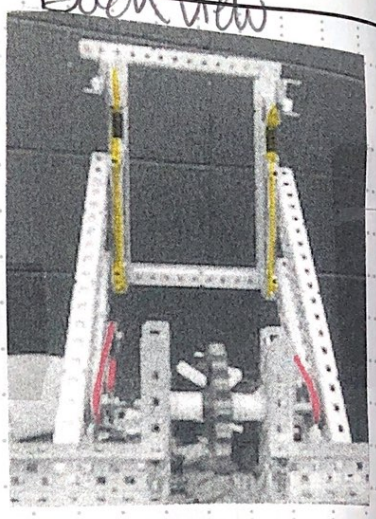
date: 9/30

Mobile Goal Lift

Side view



Back view



Pistons push up at A while the bar rotates at B to apply a downward force at C to clamp down on the mo-go.

Rubber bands later added to create more tension & help lift the Mo-go

Problem: the large Mo-GO is very heavy, causing the robot to begin to tip

Solution (10/19): Back goal lift will balance out weight

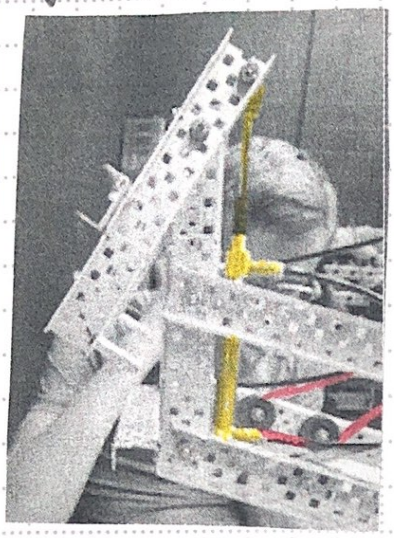
project

designed by:

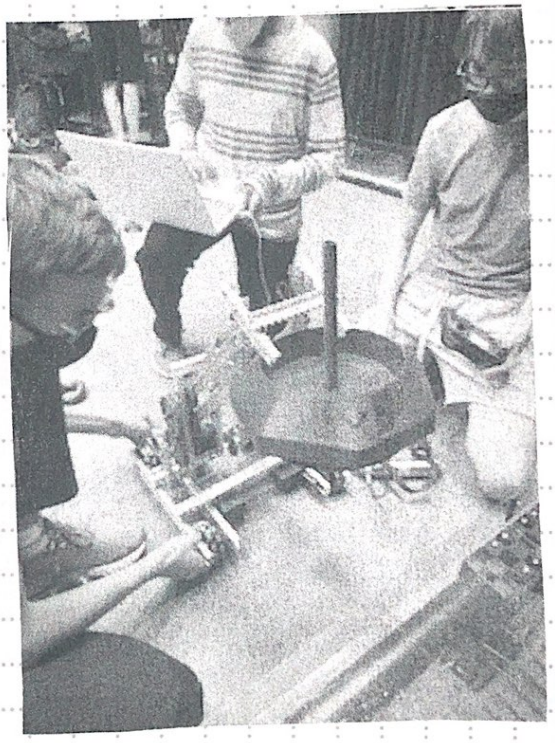
witnessed by:

date: 10/7

Pistons in extended position (Clamped)



Jacob, Mayank, and Drew working to figure out the pistons

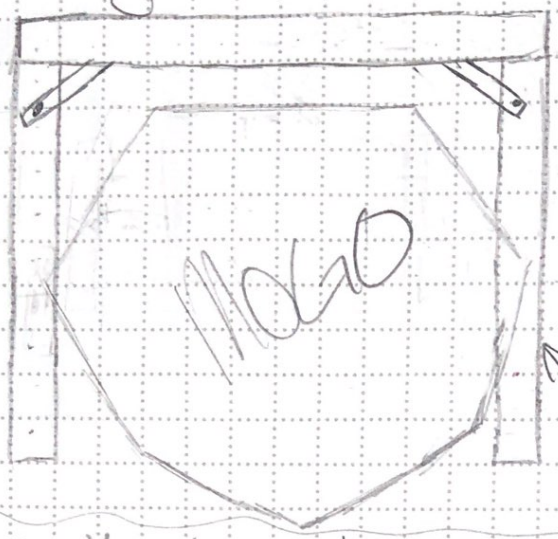


Today was spent on trouble shooting and figuring out the pistons.

project designed by: witnessed by:

date: 10/7

Skyview of back Goal lift

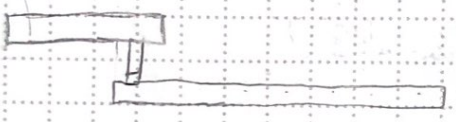


Issue: -How do we motorize
-How do we attach it to the robot?
- lift interfers with the back wheels

single C-channel

10/19

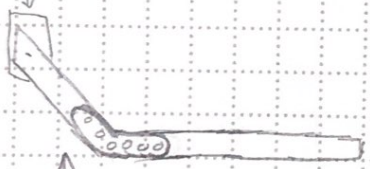
Possible replacement to clear wheels



→ Couldn't find a way to Mechanize

10/21

New approach



angle provide clearance of back wheels

project

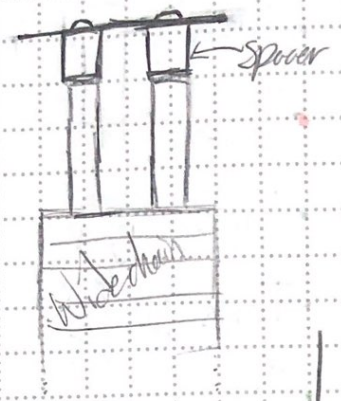
designed by:

witnessed by:

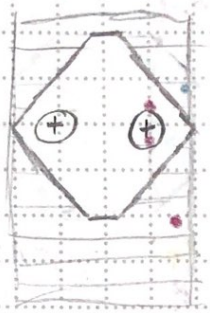
date: 10/19

Current & Updated Ring-Pegs (intake)

Front view



Sky view



★Reason for change★

↓

The intake wouldn't grab/pick up the rings

X leads to rings not sliding off X

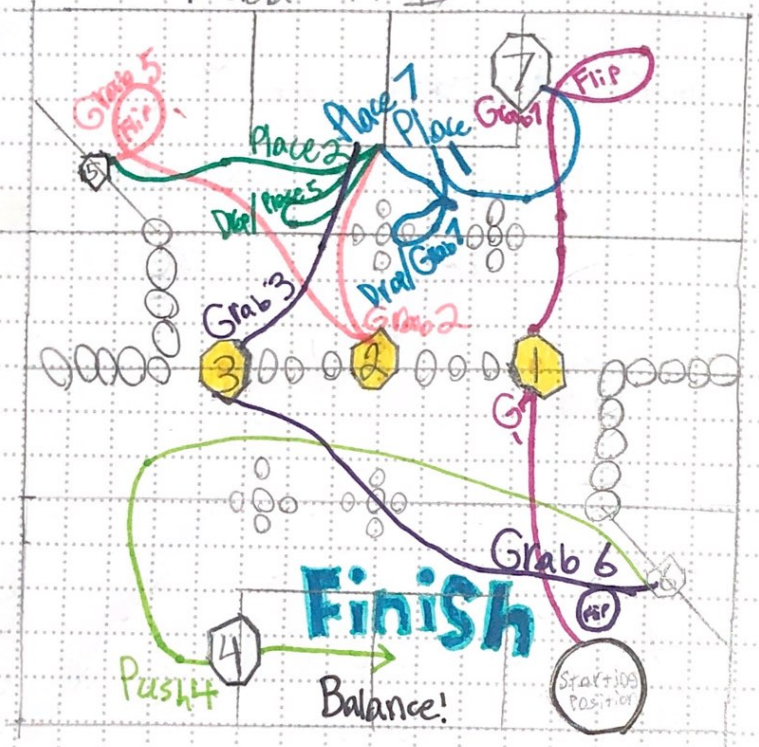
New design TBD

project designed by: witnessed by:

date: 10/19

18 Early Practice Strategy

Idea #1



- Task 1
- Task 2
- Task 3
- Task 4
- Task 5
- Task 6
- Finish

First Competition: 11/6

project

designed by: Drew

witnessed by:

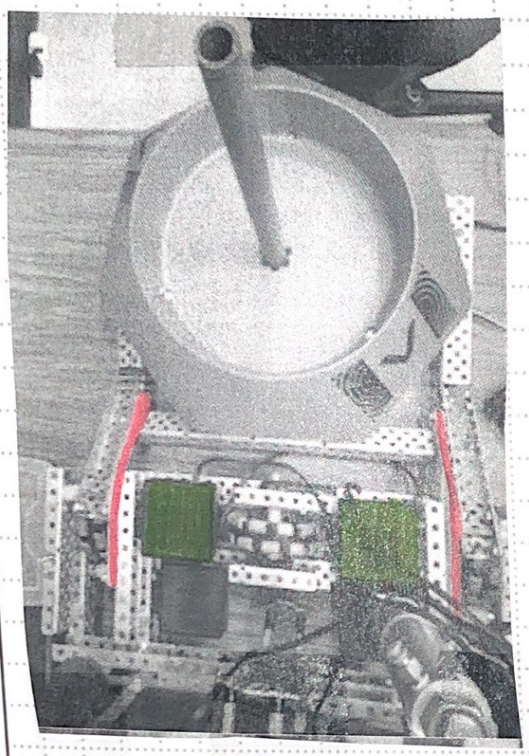
date: 10/28

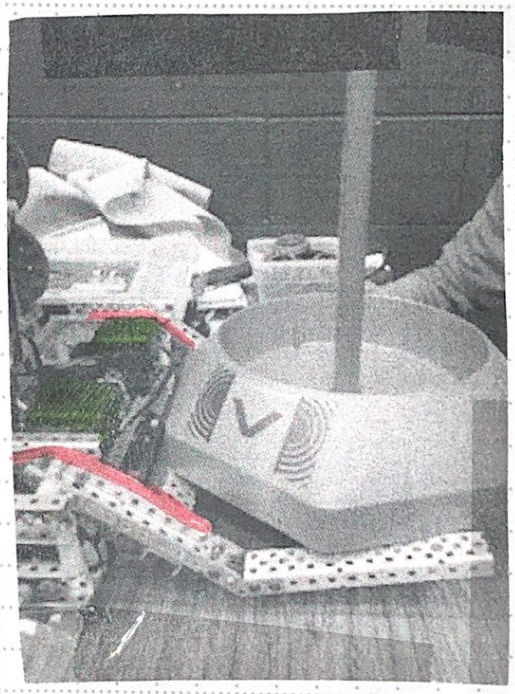
11/19

11/21

Back Mo-Go lift

The design from 10/21 seems to work better, however it was still struggling to lift the Mo-Go. Rubber bands have been added to help the motors when lifting a Mo-Go. The motor place can also be seen in the photos below.





project designed by: witnessed by:

date: 10/28

Final preps Before First Comp

This week we will be focusing on the final touches before the first competition on saturday. We have decided to exclude the ring tilt for this first competition because it is not ready & will not be ready before saturday. Drew will spend a lot of this time practicing driving.

☆ Final issue to solve before comp: Battery pack location ☆
Currently the robot is jam packed with motors, the brain, and just mechanisms in general.

project

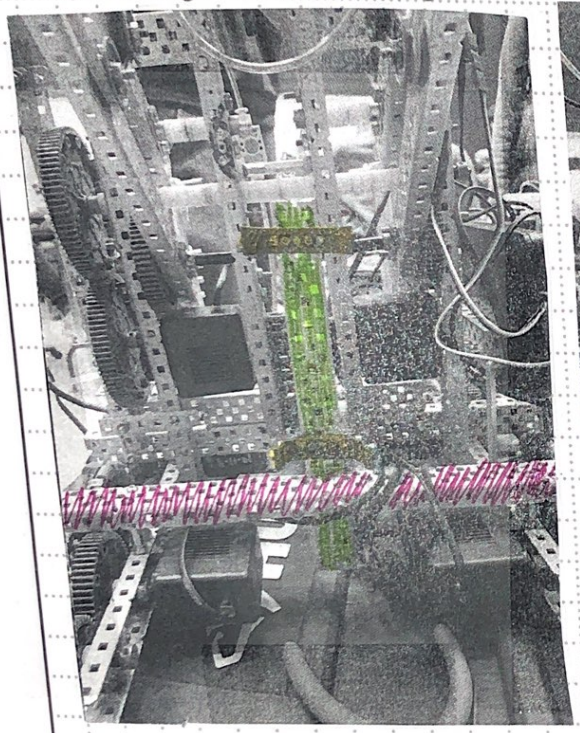
designed by:

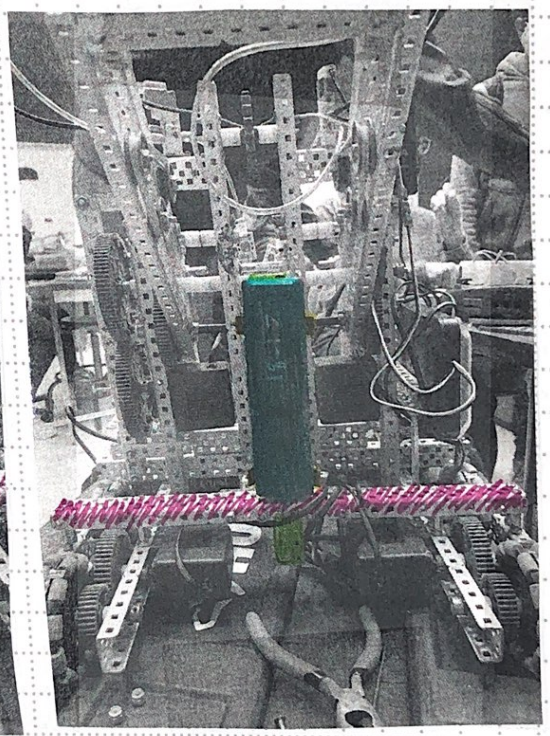
witnessed by:

date: 1/2

Battery pack location Solution:

By putting a C-channel perpendicular to the front structural support bar, and then adding Battery clips we were able to find a place for the battery without it getting in the way of our mechanisms.





Although we had be struggling to find a solution for the battery location, Drew came up with the idea of using a c-channel perpendicular to our front structural support bar. This solved the problem. We still do realize that our jam-packed robot is a problem that will need to be resolved sometime.

project Battery Pack location designed by: witnessed by:

date: 11/4

Thoughts after first Tipping point tournament

- We thought we did as expected
- Rings really are everywhere during the match
- Its chaotic
- The game can change drastically in seconds
- Rings are very important to the score
- There isn't really a complex strategy
- We don't get enough traction & drift a lot
- We want to be able to go on the ramp

Having placed 8th in the first tournament, we felt like this weekend was a great success. Having been to a VRC competition before we knew a little bit what to expect. We do wish that we had gotten more skills in. We also walked away with the Judges Award which gave us a lot of confidence.

project

designed by:

witnessed by:

date: 11/9

Problems that arose during tournament

- ramp → Our gear ratio is too high & we don't have enough torque to get up the ramp
  Approach → lower gear ration (slowed it down temporarily for the tournament)
- The cable didn't reach the battery
  ↳ This was fixed during the competition, currently our battery is up against the mag intake, but this will have to be changed once we get the intake working
- Rings would get stuck under robot
  Approach → had a channel on the front of bot to block rings from getting beneath the robot (fixed during tournament)
- Unbalanced amount of friction in tracking wheels resulted in drift throughout the competition

project

designed by:

witnessed by:

date: 11/9

Drive chain Adjustments

Since we did not have enough torque to go up the ramp the tournament we are going to change the gear ratio to 1:1. It is currently at 1.75:1.

- Another change that coincides with the gear ratio is a cut made to a central support C-channel. Prior to the cut, the gear would hit the C by removing a small 4 long side section of the C-channel, we were able clear the drive gears.

Improvements to Mo-Go Claw

Problem: "it worked, but not good enough"
- falls out when moving too quick
- couldn't get a good enough grip on Mo-Go

Changes: - 4-bar on 4-bar to allow claw part to stay parralell to the Mogo (it was at an angle before)
- more contact points between the Mo-Go + claw

↳ Didn't work. For this weekend we are sticking with previous design.

project designed by: witnessed by:

date: 11/16

Before 11-20 tournament things

- Practice on 11/16 was spent doing driving practice and programing
- The front claw has been reverted to its original design for this weekend

Current Autonomus plan (Skills + 15sec Autonomus)

- Drop 2 rings into Alliance mo-go
- Drag Alliance mo-go past the line

project designed by: witnessed by:

date: 11/18

Nov. 20th Tournament Reflection

- Went decent
- Finals alliance process went interesting
  ↳ got denied a lot
- Wish we could've gotten more skills in

Take aways

- Complete rebuild
- Focus more on skills in general

project

designed by:

witnessed by:

date: 11/23

Rebuild Plan / Goals

- room for new lift
- move drive motors back
  - more torque for 4-bar / front goal lift → Bigger Gears
  - more organized → better placement of brain, pneumatic system, etc.
- Change drive gear ratio to 1:1 to get on platform
- Vision sensor / other sensors
- having enough torque to drive over rings if stuck

Other outcomes

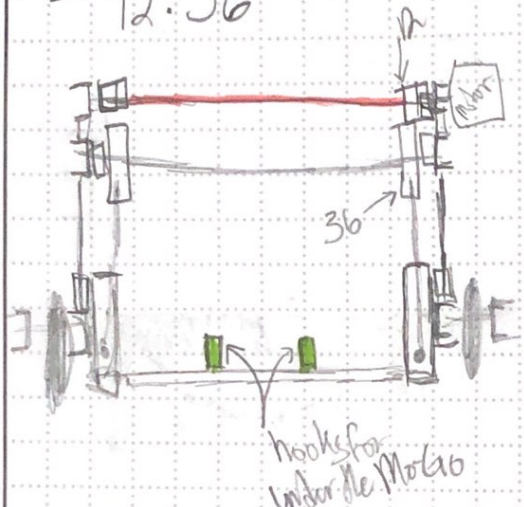
- front lift is wider as a result of widening
- changing back lift to be more consistent

project designed by: witnessed by:

date: 11/25

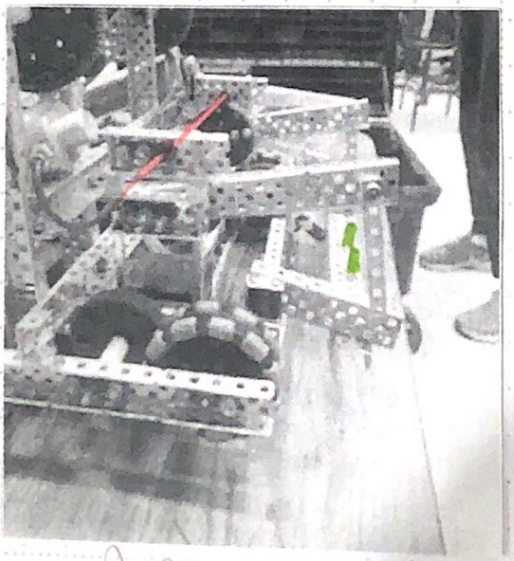
Back lift redesign

- Power both sides with a motor attatched to an axel
- 12:36

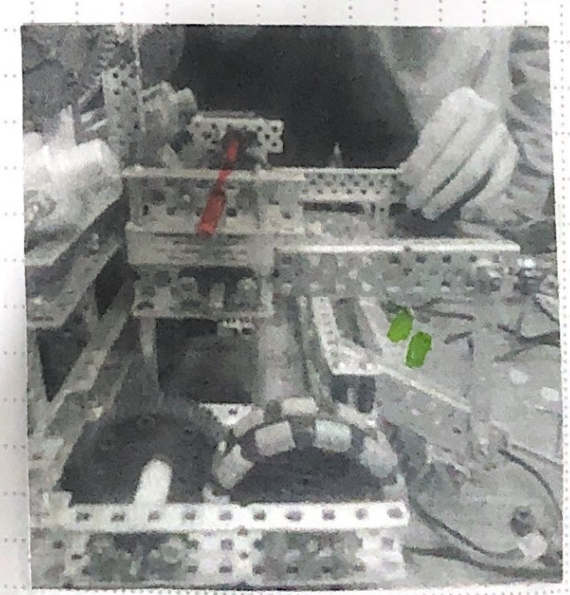


Problems

Gears start to disengage/skip because too much torque
↳ solution: Use [illegible]



Before



After

Using pictures to show the before and after, the improvements made to the back lift can be easily seen and understood. The switch to a high strenght axel was a necessary change to have enough torque to lift the Mo-Go.

project

designed by:

witnessed by:

date: 12/9

Debrief of Break Work

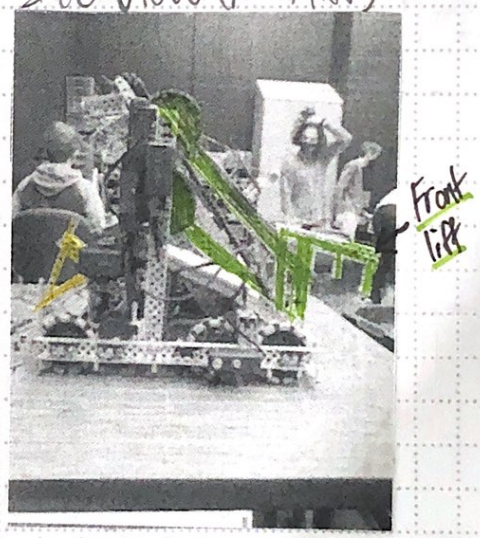
- Mechanism of back lift moved slightly (~~rubber bands~~)
- Mechanism of Front lift stayed same (~~no more rubber bands~~)

☆ Big add → ~~mg lift~~

Back view (Back lift)

Rubber bands added to help lift Mo-Go by adding tension

Side view (Front lift)

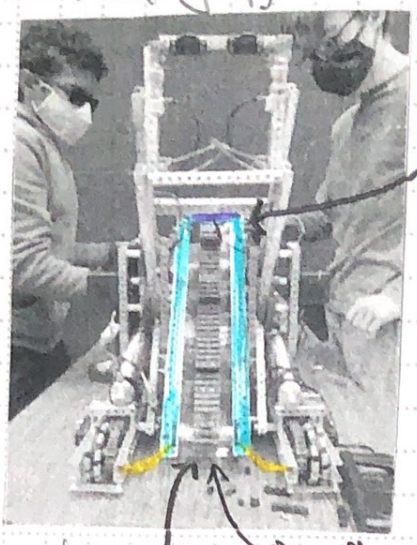


The rubber bands that were previously here to add tension are no longer needed resulting in their removal

Over winter/Christmas break, a lot a changes were made to the mechanisms that are important to the robots function to compete. The pictures shown, with use of highlight, show the changes and improvements made.

project | designed by: | witnessed by: | date: 1/4

Front view (ring lift)



↓
By having the rings hit the mesh, the bounce straight down onto the MoGo in the back lift

Use of conveyer belt to lift rings then place them into the allure mo-go pole

↓
Plexi beneath conveyer belt to allow rings to slide smoothly

- Guide rails at the opening of the ring lift not only guide the rings to the lift smoothly & the right way, but it also blocks the rings from getting stuck beneath the robot.

Pictorial of Ring lift mechanism from front view. Includes details about Guide rails, the use of mesh into our design, and how plexiglass is working to eliminate friction while the mag lift is in motion.

project

designed by:

witnessed by:

date: 1/6

Break Down of 1/8 Comp

- Finish qualifying matches rank 8
- Went into the finals w/ RamRod
- Made it all the way to, & won the finals match
- Did 3 drivers (highest: 280)
- Did 3 programing (highest: 49)

Skills

Qualifications

Elimination

Matches

① 84 60 win	② 80 122 win	③ 125 156 loss	④ 164 60 win
⑤ 80 160 win	⑥ 40 208 win		

★ Went into selection at 8th and were selected by 1115B at 7th and accepted ★

Round of 16	Quarter Finals	Semi Finals	Finals
215 100 win	81 195 win	135 174 win	110 158 win

Tournament Champions!

★ Colored match numbers = the alliance we were during said match

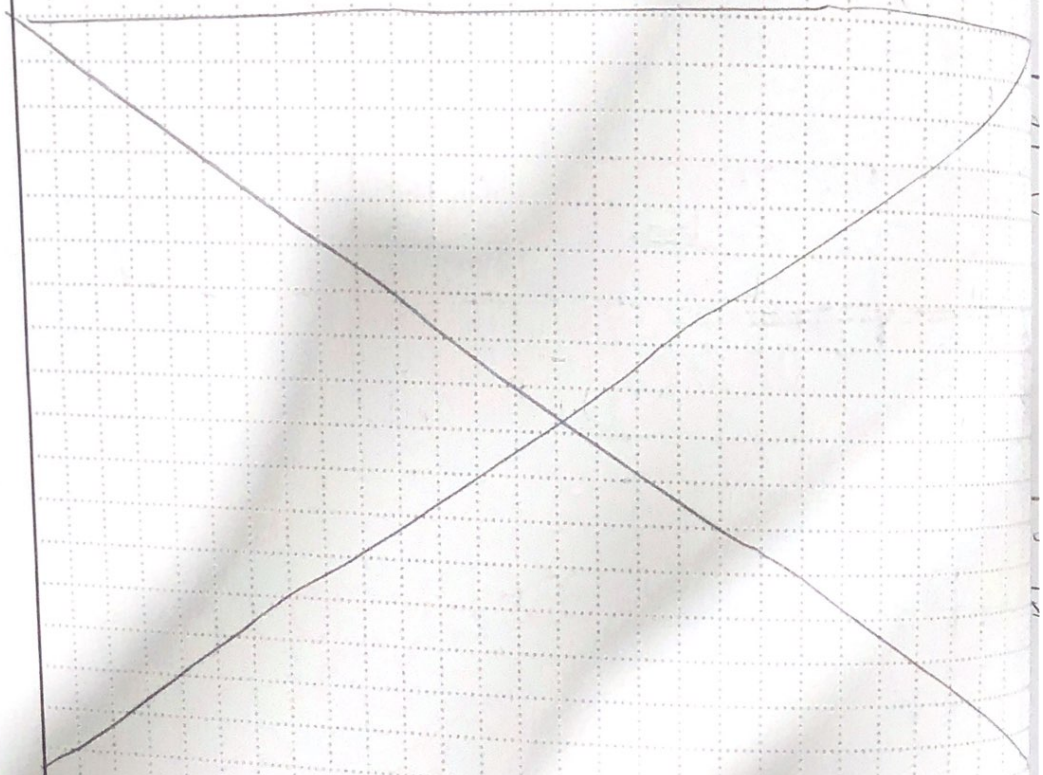
This is a break down of the January 8th tournament. This includes some of the big picture things, along with match outcomes.

project designed by: witnessed by:

date: 1/11

Changes post tournament

- Drive from 200 rpm to 300 rpm because it was too slow
- have Jacob & Olivia (maybe Mayerank) get comfortable with driving mode Drew gets the kid



This page details the changes that will be made following the 1/7 tournament.

project

designed by:

witnessed by:

date: 1/13

1/15 Comp

- Made it to 2nd elimination match
- Had a few issues regarding my intake → not being consistent enough
- Back lift issue → will be changed

Matches

Qualification

1	132	132	2	60	104	3	142	63
	win			win			win	
4	46	84	5	115	53	6	165	82
	win			loss			win	

★ Went into selection at 12th then selected by 621A and accepted ★

Elimination

Round of 16

88 ~~[illegible]~~ 61

win

Quarter Finals

178 120

loss

This contains a short debrief of the comp, along with the match outcomes & our team color is the circle around the number/match

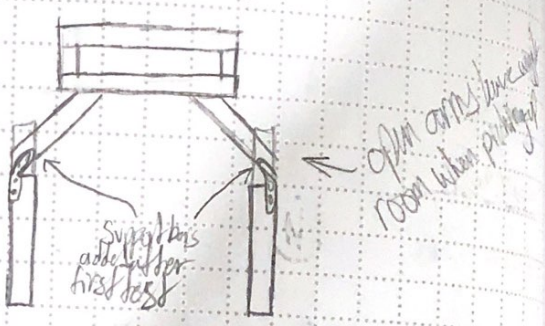
Back lift redesign

old design on page 28

- reasons for change
  - ↳ inconsistent
  - ↳ not correct height for rings
  - ↳ difficult to use

~~lift mechanism stays same~~

New Design



The back lift is being redesigned because of inconsistency in its product

project

designed by:

witnessed by:

date: 1/20

1/22 Comp

Notes

- Overall success
- Made it to Finals & won with Pigpen
- Didn't run any program skill (couldn't find time)
- Ran 3 driver skills best score was 229 (platform tipped)

Complications

- At the beginning of the day our front left wheel wasn't touching the ground. We found a temporary solution of bending one of the main chassis bars down. This will be fixed before the next comp on 1/29
- We got stuck on a ring, this was fixed comp day with stand offs under the robot to protect the wheels

New strat

- We discovered a new strategy during one of our matches, the Mo-Cao in the front arm can be held over the wall of the field to prevent it from being taken by the other team

This page details the tournament, a new strategy that was discovered, and some of the complications along with there solutions throughout the day.

project designed by: Whole team witnessed by:

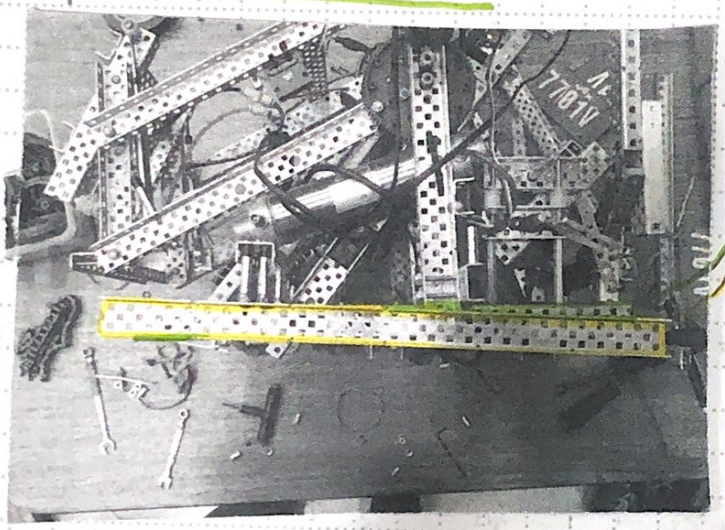
date: 1/24

1/25 Daily log

Goals

- replace left chassis bar → because of bind
- change back lift hardstop → to ensure correct placement for ring lift
- cut ~~front~~ ring lift → prevent rings from getting stuck

Replace left Chassis Bar



These two bars are supposed to be parallel, but as you can see, they are not. The yellow bar is bent upwards resulting in the front left wheel not touching the ground.

↳ the outside bar is being replaced but the inside one will not be replaced

Change Back lift hardstop

The current back lift hardstop is 2 standoffs which hit the no-Go (when its holding one). This will be changed to longer standoffs.

This daily log contains the daily goals, the changes that will be made along with the problems these changes intend to resolve.

project

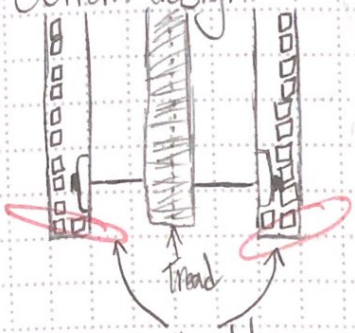
designed by: Jacob Drew

witnessed by:

date: 1/25

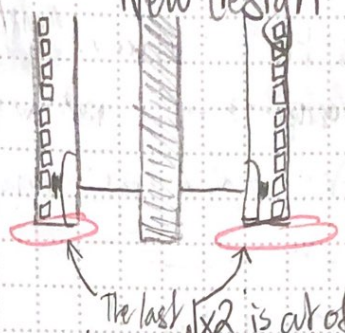
Cut ring tilt to prevent rings getting stuck

Current design



The circled areas are corners in which rings are getting stuck on when they should go straight up the tilt

New design



The last 1x2 is cut off to prevent the rings from getting stuck, they will now go straight up the tilt

Closing

Our Daily goals were accomplished as seen in each section. The following days we plan on driving practice, Programing, and final building tweeks. They may become necessary before the 1/29 tournament @ Purdue Anderson. Also added today is a PID Controllers detailed description done by Mayank, our programer. This can be seen following the book in our notebook binder.

Although Mayank had been working on this for awhile today it was finished, printed, and added to the note book.

project designed by: Drew Kachur witnessed by:

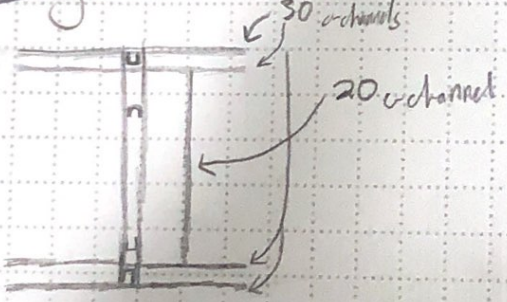
date: 1/25

1/27 Daily Log

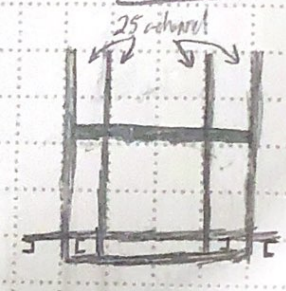
Goal

- Starfon 2nd Pinbot Chassis Design
  - ↳ The goal for this was to make a new, strong chassis that will eventually incorporate improved versions of the subsystems that are on the current robot. This was designed with stability heavily in mind due to the drive issues of our current robot.

Sky view



Front



The idea for having 4 vertical bars is to leave room for the 4-bar lift and Ring lift in-between.

Brainstorm

Use pneumatics for back McGuis lift so that drew can press a button and lift & clamp with the back lift at once. The consistency of pneumatics will also resolve the inconsistency with the ring not going onto the ring lift.

These are sketches & plans, a new Pinbot will not be constructed until a brain is available to deconstruct and reconstruct our new Pinbot. Jacek is continuing to add & design this robot, while Drew continues to drive & compete with our current robot.

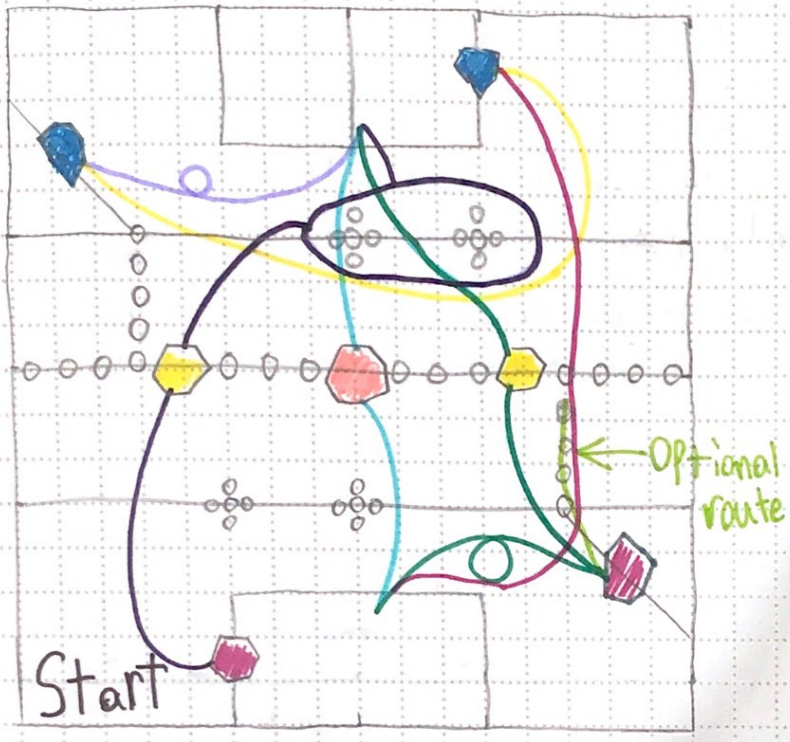
project

designed by:

witnessed by:

date: 1/27

Updated Driving Strategy (Skills)



project

designed by: Drew

witnessed by:

date: 1/28

1/29 Competition

Goals: • Make it to finals ✓ → accomplished and won the Tournament!
• Go 6-0 → didn't accomplish because of loss in first match
• Get 289 on driver skills → almost, we got 286
• have successful right auton ✓ accomplished in match 3

Matches (2 nums because opponent white screened)

(1) Pre: We will run right side auton. This should be a close match, but it will be a 2v1, we are confident
Post: We lost because of a series of issues including our platformed Mo-Go getting on scored by partner towards the end of the match. We also had trouble with our auton.

(2) Pre: Opposing team can hang so be careful. We will run right side auton. Should be a good match.
Post: Won auton but missed the 2nd yellow ★ didn't score the rings because they got stuck in partner never moved from their starting location due to technical difficulties. We won.

(3) Pre: Our partner can possess Mo-Gos, they are new bot. We will run right side auton
Post: We had total success on auton and accomplished one of our goals! Unfortunately our partner got DQed but we still got the win point.

(4) Pre: Our partner can possess 2 Mo-Gos, but no rings. Partner has left side win point. We will run right side auton. We should win this match.
Post: Rings got stuck during auton but fixed during Driver control. We didn't get the auton win, but we did win the match

(5) Pre: We will will run right side auton, our partner has a left auton win point. Partner can possess 2 Mo-Gos
Post: We won auton, but our auton didn't run at all. Our partner touched the other platform in last 30 sec but we were still able to win the match.

Note: The matches with a yellow star ★ can be found on our YouTube channel 7201V Vortex

project

designed by:

witnessed by:

date: 1/29-30

6 Pre: We will run right auton and our partner will get yellow & win point.
Post: We missed yellow in auton. We scored lots of rings. Overall was a good match and a win

Finals Selection Process Current place: 7th

Teams to Talk to about choosing us

↳ 1115B RamRod (6th) because they can possess and get rings and we would be a great team, also have auton
↳ 1115F (2nd) because we were partners in a Qualifying match and it went well

Teams to talk to about choosing them

↳ 1075B (15th) we had a match with them and it went well.

Ended up with 1075B going into elimination matches

Elimination Matches

Round of 16: Pre: Should be an easy win, we are going against a pigh and claw bot.
Post: We scored lots of rings and won the match.

Quarter Finals: Pre: Should be a good close match.
Post: There was a tug of war for most of the match between us and our opponent. The match was close for awhile

☆ Called our timeout because our drivemotors were overheating from the Tug of war, we took our robot outside to cool them down ☆

Semi finals: Pre: Will be most difficult match so far.
Post: We had another game of Tug of war. It was a game of possession. We got caught with the opponent for about 30 seconds and in the last 15 seconds of the match our opponent dropped their mo-go & it hit their platform & unscored everything.

#1 Drivers Score: 286	#2 Program Score: 29	#3 Program Score: 29	#4 Drivers Score: 229
-2 rings & all Mo-Gos Platform -Attempted the goal but missed 1 preload ring -Stuttgood Auton	-Should've gotten 69 (3 rings, 2 yellow and 1 alliance) -Activity got 3 rings & 1 yellow	-Same as #2	-The platform tipped and 2 platform Mo-Gos were unscored

SKILLS

project designed by: witnessed by:

date: 1/29-30

Finals Pre: This will be a good match all bots are from either Hamilton Heights or 2 [illegible]
Post: Good game but low scores. It was a game of possession and came down to the strenght of the bots and their claws!

Picture One Printed

The tournament was very fun and successful! Having it been at Purdue Polytech Anderson, there were so many cool machines. We got to explore in the morning prior to the drivers meeting and matches. It was so cool to see everything, it felt like we were in a museum, but with out the information signs. We ended up winning the whole tournament and we placed 4th in skills. We also went home with the judges award.

This section debriefed the tournament, gave description regarding matches, contains skills runs explanations, and contains goals regarding the tournament.

project

designed by:

witnessed by:

date: 1/29-30

Alliance Choosing: How, Who, and Why?

When it comes to selecting a team to choose or say yes to, first we ensure they they have a functioning autonomous that are compatible with ours. After that, we also prefer that they can get and score rings on Mo-Gos consistently. After choosing a few of those teams, then we take a look at their qualifying matches from earlier in the day. When looking at these we take a few things into consideration like who they played, their scores, and how they had been doing. After those processes, there are typically about 3-4 teams to choose from, we then go and talk to those teams. When talking to them, Jacob takes a close look at their robot mostly to gauge any issues they may have had during the day, drive issues and such. We then use that info to put the teams in order of who we think we would compete best with. Along with that we also have a list of teams that we are familiar with and confident in their ability to compete well.

This list includes:

- 1075B
  - ↳ Hamilton Heights, we have been at a lot of tournaments with them
- 1115B
  - ↳ Green field, Ramp Riot, we have been competing with the girl Claireon their team for a while & know their capabilities
- 9990HX
  - ↳ A team of nice people who are also good a driving
- 6842G
  - ↳ They are a typical good strong team
- 6842Z
  - ↳ great that is reliable & consistent, they make a great alliance partner
- 323V
  - ↳ Caears, we have been competing with them for awhile, they are good and consistent

project designed by: witnessed by:

date: 1/31

Panther Cornerstone League

We have decide to join a league that will take place on 2/1, 2/5, 2/8, 2/10, 2/15, & 2/22. This League has twenty-two teams. The dates above are the days in which we will attend, however there are others. We joined this league for practice, connection building, reputation building, and more skills opportunities. The match results, skills outcomes, and reflection for this will be logged in the notebook. While we are taking place in this league Jacob is continuing to Cad a second robot with more thought and planning in aspect of what has and hasn't worked for our current robot. Tonight is our first league night and we are very excited for this new experience because we have no idea what to expect.

This page is a description of what will be happening through the end of February, along with the plan of joining the Panther Cornerstone League.

project

designed by:

witnessed by:

date: 2/1

2/1 Build log

Goals

- Get drive designed
- Begin designing main MoGo lift
- Maybe Begin designing pneumatic back lift

As seen in earlier pages, we have decided to build a second robot in reflection of the outcomes of our current bot. Today we

League Night 2/1

- Prior to matches we got a few skills in

Matches

1. Great match, just barely lost because all teams are really good
   ↳ our wheel stopped working in last 15 sec
2. Complete wash had to replace motor & delay match, lose
   ↳ the motor we replaced it with was also bad
3. We just unplugged both front motors to resolve the issue temporarily
   ↳ still lost because of how slow we were
4. We were able to replace the motor with a good motor and had a good match and won

Overall, our first league night was a success! It was a lot of fun and fast pace. Luckily very little downtime. The little down time definitely was not enough to adjusted and get things done. We are really excited for the next league night!

League Night 2/5

- Another good day with alot of good teams

Matches

① Pre: Should be a good match. Our partner has an auton win point. Running Right side auton
Post: Good match, but we got stuck on ring half way through, we beat goals to the middle yellow and pulled them onto our side. Lost match with score 128-166

② Pre: Should be an easy win. We should be faster than both opponents. Same strategy as last match, both attempting win point + we go for 2 mo-bots. Jacob is going to drive.
Post: Decent match, but our auton failed to get both yellows. No win point given. Jacob struggled abit to pull rings on but was able to pull through and stack 3. won match with score 138-87

③ Pre: We are using right side auton. Gears will use left auton to get up and left yellow, were using right side to attempt at right yellow, up, then middle yellow.
Post: Pretty "schweet" match. We beat opponents to the yellow mo-go in auton, gears auton had some issues & they missed the up. Ended up winning 200-123

④ Pre: Basically a bot, we plan on running right auton & up right yellow.
Post: Decent match, our ring left got caught, ended up losing. Both autons failed. Our back left failed. Overall a rough match. 137-168

project designed by: witnessed by:

date:

2/10

Triton Central Comp Prep

- Practice/tested auton skills
- rewrote team auton
- solidified hard stops
- worked with inertial sensors

Goals for Triton Comp

- Beat our current skills scores (Both driver & programming)
- Win our elimination matches
- Win at least one award

Post Comp (2/12)

- Won Think award (auton judges award)
- ended qualifications at rank 14th
- paired with 1075B in finals
- lost in Quarter finals to the number one seed
- Placed 3rd in skills

Thoughts: We need to bring extra solenoids & pistons because our air didn't work all day. Skills was a complete bust because of air issue, however we still managed to place 3rd in skills.

Qualification

Matches

① 181 40 win	② 171 140 win	③ 168 116 loss	④ 169 175 win
⑤ 60 214 win	⑥ 117 123 win		

Went into finals alliance selection at rank 14th and ended up getting picked by 1075B at rank 13th.

Elimination

Round 16 60 131 win	Quarterfinals 180 136 loss

We plan on taking the things learned during this comp in to the next comps on our list, including state (3/12) and purdue poly tech (2/26).

project designed by: witnessed by:

date: 2/12

League Night 2/15

Goals

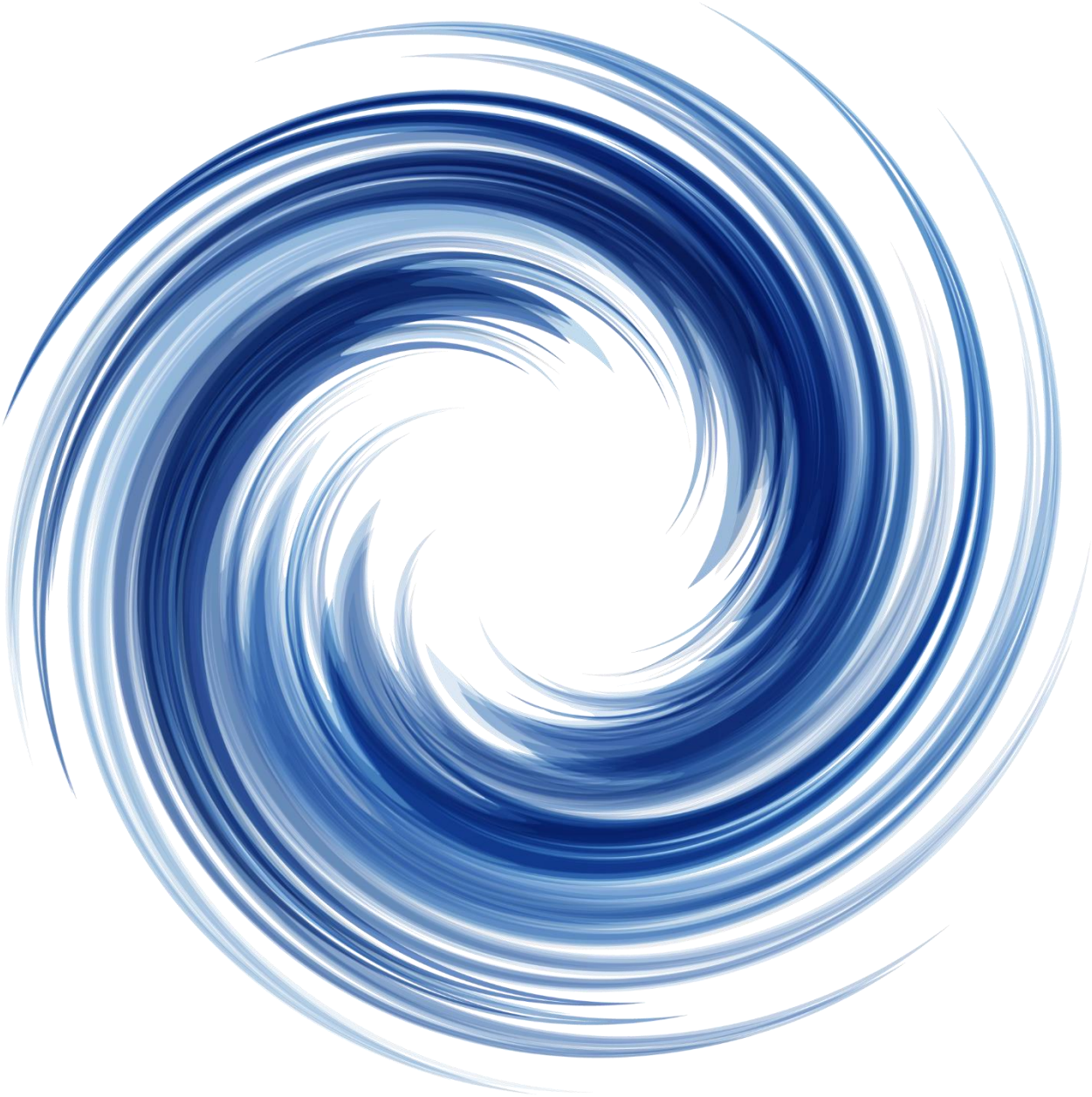
- Score 180 in auton skills
- Score 300 in driver skills
- win matches

project

designed by:

witnessed by:

date: 2/15



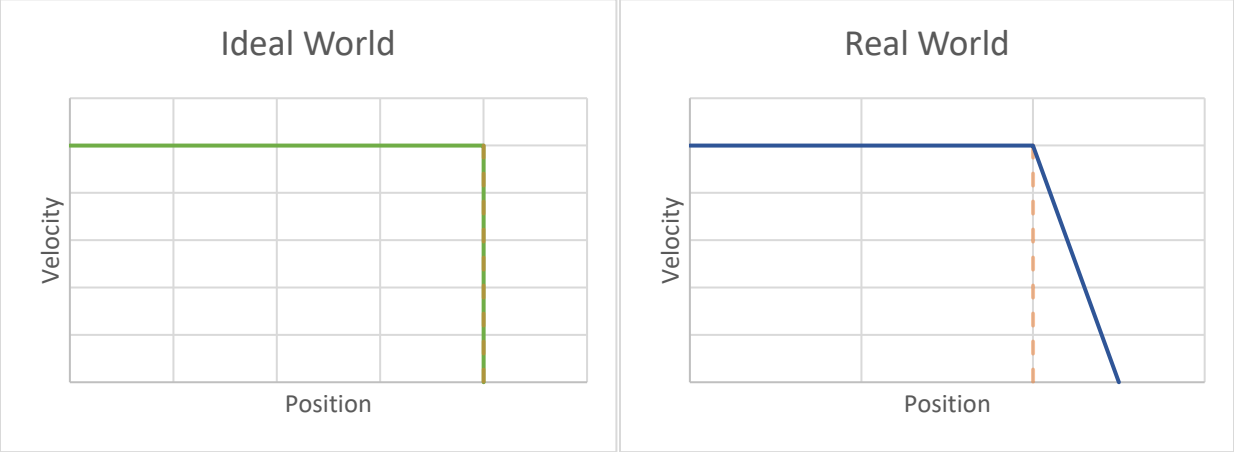
7701V | Vortex

PID Controllers and Tracking Theory

PID Controllers

Intro

The Proportional Integral Derivative (PID) Controller is a type of control system which allows for fine motor control and precise movement of the robot. In a typical robot, the robot is set to a certain speed, and it will move for that speed either for a certain amount of time or distance. This method is not very accurate and will cause the robot to be inconsistent in reaching its target. However, a PID controller will allow the robot to precisely change its speed while it is moving based on its current distance from the target position. This is necessary because there are unpredictable factors such as acceleration and friction that will affect the robot's motion and cause it to veer off track. This is shown in the following diagrams:



Proportional

The proportional component is the most important as it contributes the most to the final output power. The goal with this component is to have a large velocity when the robot is far from the target position and a small velocity when the robot is near the target position.

Error

To begin, we must first create a variable, the error. The error is the difference between the target position and the current position of the robot. For example, if the target position was 100 and the current position was 30, the error would be 70. The equation to find the error would look like the following:

$$error = targetPosition - currentPosition$$

If the robot happens to go past the target position, the error would be negative, and the robot will move backwards in the opposite direction as it attempts to reach the target

position. For instance, if the target position was 80 and the current position was 100, the error would be -20.

Output Power

To get the output power, we could just set it equal to the error, and this would work in some situations, however the output power will often be scaled incorrectly. This could cause the robot to have too little power to begin moving, or too much power to be able to stop at the target. This will cause it to get stuck in an endless loop of trying to correct itself. To fix this issue, we can multiply the error by some constant kP which can then be set equal to the output power. We will determine an acceptable value for kP later, but we can set it as 1 for now. The final equations for the proportional component would look like this:

$$\text{error} = \text{targetPosition} - \text{currentPosition}$$
$$\text{power} = \text{error} \times kP$$

Code

Currently, the code would only run once and be done. However, we do not want this to happen. Instead, the PID should be running inside of a loop which recalculates the output powers in each cycle. The code for the PID thus far would look like this:

```
void PIDMove(motor& m, double targetPosition){
    const double kP = 1;

    while(targetPosition != getCurrentPosition()){
        double error = targetPosition - getCurrentPosition();

        double power = error * kP;

        m.setVelocity(power);
    }
    m.stop();
}
```

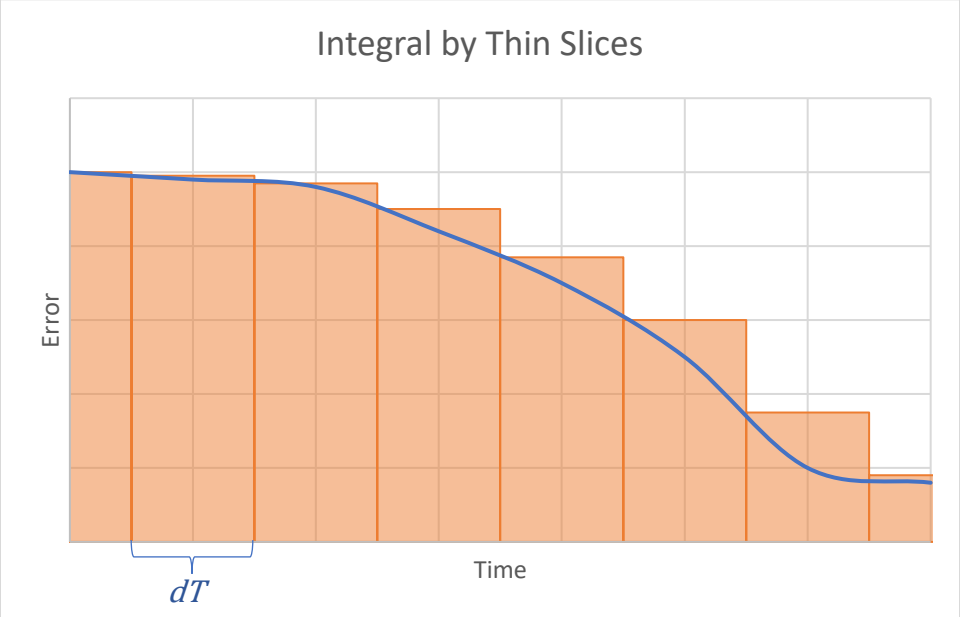
Integral

The proportional component will be enough to adjust for small disturbances, but it will not be able to fix much else. Once the robot has a small power, it will struggle to eliminate the remaining error. The integral component can fix this issue by slowly increasing the power. The integral will be looking back in the past towards all the previous errors and summing them together. In calculus, an integral is the area under a curve. Using standard calculus methods to calculate the integral will be too complicated, so instead we will be summing up the area of thin slices.

Calculations

The graph below shows a curve with an estimation of the area using thin rectangular slices. Each slice's height will be the error, and the width will be a constant dT. Then, the area of each rectangular slice can be summed to find the final area of the entire curve. It is not

perfect, but it gives a good enough estimate. A smaller value for dT will provide a more accurate estimate.



To find the area of each rectangular slice, we can multiply the width and the height, or in this case, the error and dT. The integral in each cycle of the loop is equal to the sum of all the previous areas plus the current slice's area. The equations for this would look like the following:

$$\text{area} = \text{error} \times dT$$
$$\text{integral} \mathrel{+}= \text{area}$$

Since dT is a constant amount of time ,such as 15 milliseconds, we can factor it out of the equation. Instead, we can set dT as the loop's cycle time. This means that the loop will wait for a dT amount of time before it executes the next cycle of the loop. Hence, the equation for this component will look like this:

$$\text{integral} \mathrel{+}= \text{error}$$

Consider the following case where the error is decreasing at a constant rate of 2 units per dT (not realistic). The following table shows how the integral value will change with the error:

Cycle	Error	Integral
1	15	15
2	13	28
3	11	39
4	9	48
5	7	55

Now consider what would happen if there was some external influence on the robot's motion. In the following table, the error decreases at a constant rate of 1 unit per dT:

Cycle	Error	Integral
1	15	15
2	14	29
3	13	42
4	12	54
5	11	65

As shown in the tables above, the integral was 55 after 5 cycles in the first scenario, and in the second scenario it was 65 with a slower deceleration. We can use this increasing value to add supplementary power in order to overcome the effects of the external forces.

Output Power

A higher value for the integral implies that there are some external influences on the robot causing it to slow down. To counteract this, we can add some extra power to the existing output with the integral. To account for scaling issues, we can multiply the integral by a constant like how we did with the proportional component. In this case, we will call this constant kI. The equation for output power would now look like this:

$$\text{power} = \text{error} \times kP \ + \ \text{integral} \times kI$$

Code

With the addition of the integral component, the code will now look like this:

```
void PIDMove(motor& m, double targetPosition){
    const double kP = 1;
    const double kI = 1;

    const double dT = 15;

    double integral = 0;

    while(targetPosition != getCurrentPosition()){
        double error = targetPosition - getCurrentPosition();
        integral += error;

        double power = error * kP + integral * kI;

        m.setVelocity(power);

        wait(dT);
    }
    m.stop();
}
```

Issues

The first issue with the current system occurs when the error reaches 0, i.e., the robot reaches the target position. When this occurs, the integral will be large enough to keep the

robot moving. This can be solved simply by setting the integral to 0 once the robot reaches the target position.

The second problem with this is integral windup. Integral windup occurs when the target is far away from the starting position. This will cause the integral to increase to huge values and will result in an unusable integral value at the critical moment when the robot is about to reach the target. One way to fix this issue is to set an upper limit for the integral. While this would work, there is a better solution which will allow for some more flexibility. This solution is to limit the range in which the integral is allowed to build up. This can be done by either setting a maximum error, or by setting a maximum power.

New Code

With the addition of the fixes to the issues, the new code will look like this:

```
void PIDMove(motor& m, double targetPosition){
    const double kP = 1;
    const double kI = 1;

    const double dT = 15;

    double integral = 0;

    const double maxError = 1;

    while(targetPosition != getCurrentPosition()){
        double error = targetPosition - getCurrentPosition();
        integral += error;

        if(error == 0 || getCurrentPosition() > targetPosition){
            integral = 0;
        }
        if(error > maxError){
            integral = 0;
        }

        double power = error * kP + integral * kI;

        m.setVelocity(power);

        wait(dT);
    }
    m.stop();
}
```

Derivative

Thus far we have calculated the output powers based on the current error and the past errors. With the derivative component, we will now look to the future to control the output power. The goal of the derivative component is to look at the rate of change of the error. This will show how fast the error is approaching the target, and we can decide whether to slow down or speed up. The derivative component will output a power in the direction

opposite the one that the robot is travelling in. Its magnitude is directly proportional to the speed of the robot. This component is typically outweighed by the others, but if there is some deviation from the expected movement, it will be compensated for by the derivative.

Calculations

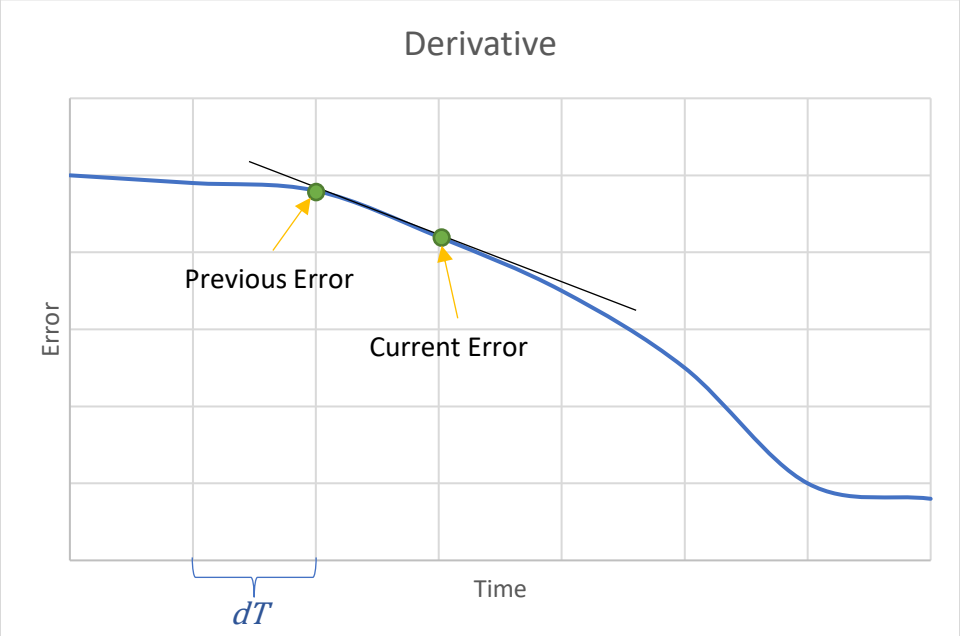
As described previously, the derivative is the rate of change or the gradient (slope) of the curve. As the curve gets steeper, the gradient will be larger. To find the derivative, we can take the slope of line tangent to the curve. In general, the equation to find the gradient looks like this:

$$\text{gradient} = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

In this case, the y-axis is the error, and the change in x is equal to dT. Substituting reveals that the equation to find the derivative is as follows:

$$\text{derivative} = \frac{\text{error} - \text{previous error}}{dT}$$

This equation is demonstrated in the graph below:



As with the integral component, dT can be ignored and factored out of the equation because it is a constant amount of time. To get the previous error, we need to set it equal to the current error after we calculate the derivative in each cycle of the loop. The final equations will look like the following:

$$\text{derivative} = \text{error} - \text{previous error}$$
$$\text{previous error} = \text{error}$$

Output Power

Like the proportional and integral components, we need to create a constant kD so we can properly scale the output powers. A value that is too high will completely overpower the other two components, meanwhile a value that is too low makes the component unnecessary. To incorporate it into the output power calculation, we can add it as follows:

$$\text{power} = \text{error} \times kP \ + \ \text{integral} \times kI + \text{derivative} \times kD$$

Code

The final code with all three components added will look like this:

```
void PIDMove(motor& m, double targetPosition){
    const double kP = 1;
    const double kI = 1;
    const double kD = 1;

    const double dT = 15;

    double integral = 0;
    double prevError = 0;

    const double maxError = 1;

    while(targetPosition != getCurrentPosition()){
        double error = targetPosition - getCurrentPosition();
        integral += error;

        if(error == 0 || getCurrentPosition() > targetPosition){
            integral = 0;
        }
        if(error > maxError){
            integral = 0;
        }

        double derivative = error - prevError;
        prevError = error;

        double power = error * kP + integral * kI + derivative * kD;

        m.setVelocity(power);

        wait(dT);
    }
    m.stop();
}
```

Tuning

The most important and difficult part of creating a PID controller is tuning the constants. There are several methods of accomplishing this, but the two most common are trial-and-error and mathematical tuning. While tuning the constants, it is important to keep track of the error and the sensor values instead of just going by what appears to be working.

Factors

There are 5 main factors that determine the behavior of the PID controller. They are as follows:

- Rise time – The time it takes the robot to go from the starting position to the target
- Overshoot – How far the robot travels beyond the target
- Settling time – The time it takes the robot to settle down after it encounters a disturbance
- Steady-state error – The error when the robot is at equilibrium
- Stability – The smoothness of the robot's motion

The following table shows how each factor is affected by an increase in the constants:

Parameter	Rise Time	Overshoot	Settling Time	Steady-State Error*	Stability
kP	Decrease	Increase	N/A	Decrease	Worsens
kI	Decrease	Increase	Increase	Decrease	Worsens
kD	N/A	Decrease	Decrease	N/A	Improves**

* For the steady-state error, it is important to note that reliability is more important than accuracy. If you are always 5 units past the target, you can easily adjust for that, but if the steady-state error varies between $\pm$ 2.5 units, it is much more difficult to account for it.

** This only applies if kD is small enough. If it is too large, it can have the opposite effect and worsen the stability. This happens because the derivative component acts in the opposite direction of the proportional and integral components. This could cause the robot to stop prematurely before reaching the target, or even to start travelling in the opposite direction of the target. Because the derivative component is proportional to the robot's speed, it will weaken when the robot slows down. After this occurs, the other components will add more power causing the derivative to strengthen again. This loop will continue and cause the motion of the robot to appear jittery and uncontrolled.

Manual Tuning

This method of tuning the constants can be very repetitive and tedious, but it is widely regarded as the best method. This method can allow the robot to be tuned for each specific use case.

The first step is to set all the constants to 0 as a starting point. This effectively disables each component. Then, each constant will be adjusted one after another. This typically happens in the order of proportional, derivative, and then integral. This process relies on setting a predicted value then adjusting based on its performance.

1. To begin, kP must be increased until the robot oscillates slightly, just once or twice. The goal is to reach the target quickly without being too violent with the motion. The robot then needs to settle in a reasonable amount of time.

2. The next step is to increment kD until the steady-state error is decreased to a reasonable amount. This will allow the robot to maintain it's fast motion while minimizing overshoot at the same time. It may be necessary to go back and adjust kP slightly.
3. Next, kI must be increased until any other minor disturbances are removed. It is probably necessary to revisit the previous steps to adjust for small variances.
4. Using the factors table, the constants can then be adjusted further until the robot has a swift, smooth, and reliable motion.

Mathematical Tuning

Another method to tune a PID is the mathematical method. This method can be used to get a good estimate for the starting values, but some manual tuning will still be required afterwards.

Like the trial-and-error method, we must begin by disabling all 3 components by setting all the constants to 0.

1. First, kP needs to be increased until the robot oscillates continuously. These oscillations must be stable and consistent. This value can then be recorded as the ultimate gain or kU.
2. Then, the period of the oscillations must be measured. The period is the time it takes the robot to move through a full cycle back to the starting point. This can be done either through programming or with a stopwatch. A more accurate measurement for the period can be acquired by taking the average of many periods. This can be recorded as the period for ultimate gain or pU.
3. The final step with mathematical tuning is to approximate the constant values based on the following table:

Controller Type	kP	kI	kD
P	$0.5 \times kU$	0	0
PI	$0.45 \times kU$	$0.54 \times kU \div pU$	0
PD	$0.8 \times kU$	0	$0.1 \times kU \times pU$
PID	$0.6 \times kU$	$1.2 \times kU \div pU$	$0.075 \times kU \times pU$

As mentioned previously, it is important to adjust the constants further using the manual method in order to achieve perfect motion.

Conclusion

This concludes the PID section of this document. We have looked at the math behind each component and implemented them accordingly. It is important to note that not all three components are needed for all applications. Sometimes, it may better to use a PI or a PD controller instead. In the following sections of the document, we will be looking at how a PID controller can be used in conjunction with a position tracking system to achieve a more consistent autonomous.

Position Tracking

Intro

The Absolute Positioning System (APS) is a system that keeps track of the absolute position (cartesian coordinates) of the robot during a match. It allows us to tell the robot to go to a certain coordinate instead of just going forward a certain distance.

Units

- Distance – Inches
  - The field and game elements are measured in inches, so it would be more convenient to use inches
- Angles – Radians
  - Radians are preferred over degrees because they are the native output of the algorithm and the native input for built-in trigonometric functions
- Time – Seconds or Milliseconds

Definitions

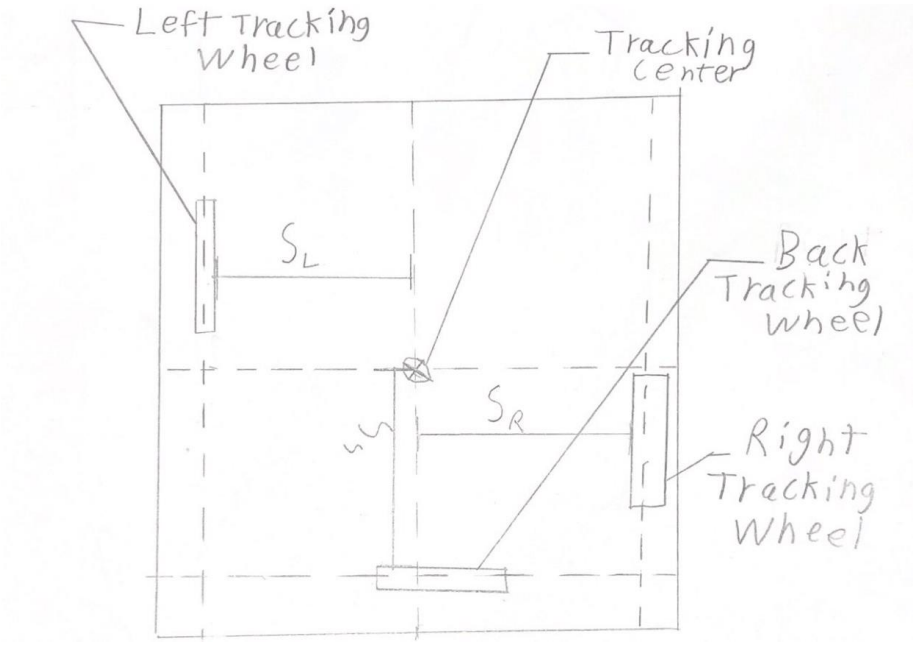
- Tracking center – The point which will be tracked, preferably the center of rotation
- Tracking wheels – The auxiliary wheels with rotation sensors (encoders) attached that are used to track the one-dimensional displacement of each wheel
  - There are three of them, one on each side and one which is rotated to be perpendicular to the other two
- Inertial sensor – A type of gyro sensor which can be used to find the robot's heading
- S_L – Left-right distance from the tracking center to the left tracking wheel
- S_R – Left-right distance from the tracking center to the right tracking wheel
- S_S – Forward-backward distance from the tracking center to the back tracking wheel
- $\overrightarrow{d_0}$ – The previous global position vector
- $\overrightarrow{d_1}$ – The current global position vector
- θ_0 – The previous global orientation
- θ_1 – The current global orientation

Tracking Theory (Odometry)

Odometry is the core of the position tracking system. It uses a predefined point called the tracking center to update the position vector $\vec{d}$ and the orientation vector θ in real-time.

Placement of Tracking Wheels

The diagram below shows a sample robot chassis with wheel placement:



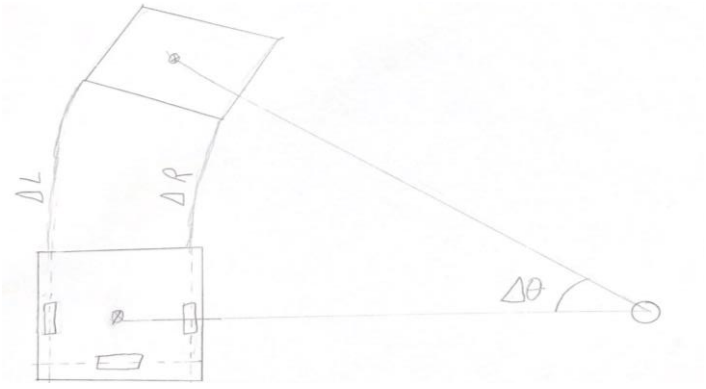
In this example, the tracking center is located at the center of the chassis which also happens to be the center of rotation. However, the tracking center does not need to be located at the center of rotation, but this will make it easier to develop the motion algorithms later.

The tracking wheels can be placed anywhere along the dotted lines without affecting the calculations. The important figures here are the perpendicular distances from the tracking wheels to the tracking center.

Change in Wheel Position

The central algorithm of the position tracking system works by simulating the motion of the tracking center as an arc over an infinitely short time interval. For this example, we will assume that the robot does not deviate from this arc and that the tracking wheels are aligned with the center of the robot. Later, we will see that neither of these are necessary. Both the left and right wheel paths are concentric arcs with the tracking center's arc.

The figure below shows a sample motion that the robot might make:



From the diagram above, we can find a few important measurements. First, we can calculate the change in wheel position for each tracking wheel. This can be done with the following equation:

$$\Delta W = \frac{2\pi r\ (E_1 - E_0)}{360}$$

In this equation, ΔW is the change in wheel position, r is the radius of the tracking wheel, E_1 is the current encoder position and E_0 is the previous encoder position. First, we need to calculate the circumference of the tracking wheel. This is done by multiplying the radius by 2π. Next, we need to find the section of the wheel which has moved. This can be done by dividing the change in encoder positions by 360. This is necessary because the sensors' output values are in degrees. Finally, we can multiply these two values together to find ΔW.

Orientation with Arcs

Another element that we can derive from the diagram above is the angle of the arc, which happens to be the same as the change in orientation of the robot, $\Delta\theta$. This angle can be calculated using the arc length formula:

$$s = \theta \times r$$

In the equation above, s is the arc length, r is the radius of the arc, and θ is the central angle. We can apply this equation to the robot's movement as such:

$$\begin{aligned} \Delta L &= \Delta\theta \times r_L \\ \Delta L &= \Delta\theta\ (r_A + S_L) \\ \frac{\Delta L}{\Delta\theta} &= r_A + S_L \\ r_A &= \frac{\Delta L}{\Delta\theta} - S_L \end{aligned} \qquad \begin{aligned} \Delta R &= \Delta\theta \times r_R \\ \Delta R &= \Delta\theta\ (r_A - S_R) \\ \frac{\Delta R}{\Delta\theta} &= r_A - S_R \\ r_A &= \frac{\Delta R}{\Delta\theta} + S_R \end{aligned}$$

These two equations can then be combined while also eliminating the unknown value r_A, the tracking center's arc's radius:

$$\begin{aligned} \frac{\Delta L}{\Delta\theta} - S_L &= \frac{\Delta R}{\Delta\theta} + S_R \\ \Delta L - S_L \times \Delta\theta &= \Delta R + S_R \times \Delta\theta \\ \Delta L - \Delta R &= S_L \times \Delta\theta + S_R \times \Delta\theta \\ \Delta L - \Delta R &= \Delta\theta\ (S_L + S_R) \\ \Delta\theta &= \frac{\Delta L - \Delta R}{S_L + S_R} \end{aligned}$$

Orientation with Inertial Sensor

An alternative to calculating the change in orientation with arcs is to do it with a gyroscope such as the V5 inertial sensor. The V5 inertial sensor can read the heading of the robot in a

much more compact way than tracking wheels. This method has a few advantages as well as drawbacks:

Pros	Cons
• The robot will know its orientation regardless of where it is • The robot will keep recording the orientation even if the tracking wheels lose contact with the ground • One of the side tracking wheels is no longer necessary; only one of the side wheels and the back wheel are needed • The inertial sensor takes up much less space than a tracking wheel	• It is significantly less accurate to use the inertial sensor over the tracking wheels for the orientation ○ One experiment found that the inertial sensor was inaccurate by over 22%, meanwhile the tracking wheels were only inaccurate by 0.04% • The inertial sensor is prone to drift and noise, i.e., the values will be changing even if the robot is not moving • The inertial sensor must be calibrated frequently which can be time consuming

The equation for finding the change in orientation with the inertial sensor is this:

$$\Delta\theta = \text{current heading} - \text{previous heading}$$

Regardless of which method is used, the value will be approximately the same. In addition, since any translation without a rotation of the tracking center does not change the difference between ΔL and ΔR, this equation can be used anywhere on the field. It does not matter whether the robot did a swerve turn or an in-place turn, the calculations will be the same. This also means that the orientation can be calculated as an absolute quantity instead of it being relative to the robot:

$$\theta_1 = \theta_0 + \Delta\theta$$

Change in Local Position

Using the change in orientation, the translation of the robot can also be calculated. To do this, we need to first find the radius of the tracking center's arc:

$$r_A = \frac{\Delta R}{\Delta\theta} + S_R$$

This can be done using either the left or right tracking wheels. Again, the motion of the robot can be modelled as an arc. The y-coordinate can be calculated as the chord length of this arc:

$$\Delta\overrightarrow{d_{ly}} = 2\sin\frac{\Delta\theta}{2} \times \left(\frac{\Delta R}{\Delta\theta} + S_R\right)$$

The robot will often stray from this arc resulting in additional translation. This can be calculated using a second arc which is perpendicular to the first. This chord length of this arc will be representative of the x-coordinate. It can be calculated in a similar manner to the y-coordinate:

$$\Delta\overrightarrow{d_{lx}} = 2\sin\frac{\Delta\theta}{2} \times \left(\frac{\Delta S}{\Delta\theta} + S_S\right)$$

Then, the global position vector can be calculated by summing all the previous local position vectors.

These equations will currently only work if $|\Delta\theta| > 0$. Otherwise, it will result in a divide by zero error. To fix this, we can add a conditional statement if $\Delta\theta \neq 0$:

$$\Delta\overrightarrow{d_l} = \begin{bmatrix} 2\sin\frac{\Delta\theta}{2} \times \left(\frac{\Delta S}{\Delta\theta} + S_S\right) \\ 2\sin\frac{\Delta\theta}{2} \times \left(\frac{\Delta R}{\Delta\theta} + S_R\right) \end{bmatrix}$$

Otherwise, we can just set the local position vector equal to S_S and S_R respectively:

$$\Delta\overrightarrow{d_l} = \begin{bmatrix} S_S \\ S_R \end{bmatrix}$$

Calculating Global Position

To calculate the global position vector, we need to rotate the local position vector by the average orientation, θ_m. We can find the average orientation using the following equation:

$$\theta_m = \theta_0 + \frac{\Delta\theta}{2}$$

To rotate the local position vector, we can use one of several methods. The simplest way is to convert the vector to polar coordinates, decrement the angle by the average orientation, then convert the vector back to rectangular coordinates. We can convert the vector to polar coordinates with the following equations:

$$r_p = \sqrt{\Delta\overrightarrow{d_{lx}}^2 + \Delta\overrightarrow{d_{ly}}^2}$$

$$\theta_p = \tan^{-1}\frac{\Delta\overrightarrow{d_{ly}}}{\Delta\overrightarrow{d_{lx}}}$$

Then, the angle can be rotated by subtracting the average orientation:

$$\theta_p \mathrel{-}= \theta_m$$

Now, the change in local position vector can be converted back to rectangular coordinates to find the change in global position vector, $\overrightarrow{\Delta d}$:

$$\Delta\vec{d} = \begin{bmatrix} r_p \cos\theta_p \\ r_p \sin\theta_p \end{bmatrix}$$

Finally, the absolute global position can be calculated by adding the change in position vector to the previous global position vector:

$$\overrightarrow{d_1} = \overrightarrow{d_0} + \Delta\vec{d}$$

Motion Algorithms

The position tracking system can be used in conjunction with PID controllers to achieve a smooth motion to go to specified coordinates. However, the PID controller will only work in one dimension as it currently is. To fix this, we can convert a coordinate into a distance from the robot's position. The robot will first turn to face the point and then it will travel the distance towards it. The equation for finding angle is the following:

$$\theta_t = \tan^{-1}\frac{\overrightarrow{d_{1y}} - y_t}{\overrightarrow{d_{1x}} - x_t}$$

In this equation, x_t and y_t are the target coordinates, and θ_t is the global angle that the robot needs to turn to so that it can be facing the target point. Then, the distance the robot needs to travel can be found by taking the hypotenuse of the change in position:

$$d = \sqrt{\left(\overrightarrow{d_{1y}} - y_t\right)^2 + \left(\overrightarrow{d_{1x}} - x_t\right)^2}$$

These two values can then be used as the target value for the PID function in order to efficiently and accurately arrive at the target position.

Code

The position tracking system can be implemented with the following code:

```
//Position Tracking
namespace PositionTracking{
  bool usingInertial = false;

  //radius of tracking wheel (in inches)
  const double WHEEL_RADIUS = 1.6125;

  //left-right distance between left or right tracking wheel and tracking
center (in inches)
  const double s_L = 6;
  const double s_R = 6;
  //forward-backward distance between back tracking wheel and tracking center
(in inches)
  const double s_S = 0.875;

  double pos_x;       //x position of robot (in inches)
  double pos_y;       //y position of robot (in inches)
  double pos_theta;   //angle of robot's heading (in radians)
```

```
  //main position tracking function, will be used in its own thread
  void positionTracking(){
    //previous position of encoders: L = left, R = right, S = back
    double encL_0 = 0, encR_0 = 0, encS_0 = 0;

    //previous positions of robot: theta = heading angle (in radians), x & y
= position (in inches)
    double theta_0 = 0, d_0_x = 0, d_0_y = 0;

    while(true){
      uint32_t timeStart = Brain.Timer.system();
      //encoder sensor readings (in degrees)
      double encL = leftRotationSensor.position(deg);
      double encR = rightRotationSensor.position(deg);
      double encS = backRotationSensor.position(deg);

      //Change in position for each tracking wheel
      //Equation is circumfrence of wheel times section of circle where the
encoder's angle changed:
      //delta = 2 * pi * radius * deltaEnc / 360
      double delta_L = 2 * M_PI * WHEEL_RADIUS * (encL - encL_0) / 360;
      double delta_R = 2 * M_PI * WHEEL_RADIUS * (encR - encR_0) / 360;
      double delta_S = 2 * M_PI * WHEEL_RADIUS * (encS - encS_0) / 360;

      //set previous encoder angles to current ones for use in next loop
cycle
      encL_0 = encL; encR_0 = encR; encS_0 = encS;

      double theta_1, delta_theta;

      //get robot heading from inertial sensor and convert it from degrees to
radians
      if(usingInertial){
        theta_1 = inertialSensor.rotation(deg) * M_PI / 180;
        delta_theta = theta_1 - theta_0;
      }
      //get robot heading from tracking wheels instead because it is much
more accurate
      else{
        delta_theta = (delta_L - delta_R) / (s_L + s_R);
        theta_1 = delta_theta + theta_0;
      }

      //change in local position
      double delta_d_l_x = 0, delta_d_l_y = 0;
      //if delta theta is 0, set change in local position to delta s and
delta r; this avoids a divide by zero error
      if(delta_theta == 0){
        delta_d_l_x = delta_S;
        delta_d_l_y = delta_R;
      }
      else{ //otherwise set it to this:
        delta_d_l_x = 2 * std::sin(delta_theta / 2) * (delta_S / delta_theta
+ s_S);
        delta_d_l_y = 2 * std::sin(delta_theta / 2) * (delta_R / delta_theta
+ s_R);
```

```
      }

      //calculate average orientation
      double theta_m = theta_0 + delta_theta / 2;

      //convert change in local position to polar coords so we can rotate it
      double pl_r = std::sqrt(delta_d_l_x * delta_d_l_x + delta_d_l_y *
delta_d_l_y);
      double pl_theta = std::atan2(delta_d_l_y, delta_d_l_x);

      //rotate the theta of the polar coord by -theta_m
      pl_theta -= theta_m;

      //convert back to rectangular coords to find change in position
      double delta_d_x = pl_r * std::cos(pl_theta);
      double delta_d_y = pl_r * std::sin(pl_theta);

      //set global position to previous pos + change in pos
      double d_1_x = d_0_x + delta_d_x;
      double d_1_y = d_0_y + delta_d_y;

      //set previous positions as current positions for use in next loop
cycle
      d_0_x = d_1_x; d_0_y = d_1_y; theta_0 = theta_1;

      //assign the global positions to the vars located outside the function
so they can be used elsewhere
      pos_x = d_1_x; pos_y = d_1_y; pos_theta = theta_1;

      //sleep this thread for 15 milliseconds to conserve power and get more
precision
      this_thread::sleep_until(timeStart + 15);
    }
  }

  //functions that can be used externally to get the positions

  //flipped x and y to account for encoder reversing
  double x(){return pos_y;}
  double y(){return pos_x;}
  double theta(){return pos_theta;}

  //returns angle with wrap
  //range ? (-pi, pi) : (0, 2pi)
  double thetaWrapped(bool range){
    //-pi to pi
    if(range){
      double temp_theta = fmod(theta() + M_PI, 2 * M_PI);
      if(temp_theta < 0) temp_theta += 2 * M_PI;
      return temp_theta - M_PI;
    }
    //0 to 2pi
    double temp_theta = fmod(theta(), 2 * M_PI);
    if(temp_theta < 0) temp_theta += 2 * M_PI;
    return temp_theta;
  }
}
```

Conclusion

The position tracking system uses encoder sensors placed on tracking wheels to calculate the absolute position of the robot. This allows the robot to always know its locational information. It can be used in conjunction with PID controllers to achieve smooth, articulate, and accurate motion. This system will make programming autonomous routines significantly easier and more consistent.