Programming Notebook

4862A

Team Number

Ravens

Team Name

Ashbury College

School

04/29/2023 — Start Date

02/16/2024 — End Date

2 of 2 — Book #

VEX ROBOTICS

REC FOUNDATION

Table of Contents

Pre-season Logs

2023/04/29 @home

Programming log #1

finished point to point rudimentary algorithm.
Calculates turn angle(starting from 0 at y-axis) needed and wheel rotation needed to travel to a point, from a point.

2023/04/30 @home

Programming log #2

Worked on odometry, finished the secondary odometry program I had previously started which was based on team 5225a E-Pilons documentation. Looked over their code to gain a better understanding of why certain computations were made the way they were. Geometric proof below:

Also figured out that I needed to add some end calculations to calculate global variables instead of only calculating local variables.

Name: Carol Wang

2023/05/01 @home

Programming log #3

Worked out some kinks in the motion profile algorithm. The program was using outputting much higher velocity than it was supposed to. Went through each line of logic individually to figure out where the miscalculation was occurring. The problem was that a major variable was wrong, causing the program to calculate with max velocity instead of Vm. Also, motion profile was set to the wrong motor configuration(200 RPM instead of 600 RPM gearing).

2023/05/04 @home

Programming log #4

Tested the corrected motion profile and it worked. Tested it at 4 inch, 8 inch, and 12 inch distances driving straight, three trials each, and all trials were within ¼ inch error.

Name: Carol Wang

2023/05/07 @home

Programming log #5

Odometry research. I need to find a way to eliminate error in my odometry program. Right now, error is small when using a motion profile, but big when moving the robot by hand, meaning wheel slippage is happening. Theoretically, implementing motion profiles all throughout the autonomous would reduce odometry error, but if outside factors, like game objects thrown by opponents, cause the robot to shift, jerk error will still happen. This means the best solution would be implementing dead wheels. I needed to research whether it is worth the time to implement dead wheels(skill issue does not use them and they held the world record consistently last year) or whether a motion profile would be enough.

drive done
final-x: -0.5614231 , final-y: 13.77126
drive done
final-x: 0.1283911 , final-y: 12.49185
drive done
final-x: 0.1483151 , final-y: 12.50186
off by around 1 inch, which tracks because its around 9.24125476017 off, which is error of theta when turning by hand slowly
-> switched to motion profile tests
final-x: 0.03064174 , final-y: 11.61481 - 11.5
final-x: 0.03588512 , final-y: 11.56992 - 11.5
final-x: 0.04277862 , final-y: 11.68656 - 11.65
final-x: 0.044975 , final-y: 11.66865 - 11.75

final-x: 0.06308313 , final-y: 23.66943
final-x: 0.01251391 , final-y: 23.66055
final-x: 0.1064913 , final-y: 23.60634
0.4 + 1/6 inch off

1 - around 1/4 to 1 inch off(left vs right side), odom says 3 inches
1.5 inch overshoot, odom overshoot by 3
1.6 inch overshoot

final-x: 1.559357 , final-y: 32.8167
final-x: 0.08805264 , final-y: 35.83166
final-x: 0.03916624 , final-y: 35.81395
Changed 4 to 4.2

Name: Carol Wang

Programming log #5 continued

To continue my research, I used ChatGPT to find optimal solutions to questions I had:

Should I use motion profiling or dead wheels to reduce slip error in odometry?

How do I limit error in my odometry implementation?

- Use high quality sensors
- Implement sensor fusion(such as encoders and gyros)
- Calibrate your sensors(this involves measuring sensor's output under known conditions and adjusting the readings to match the expected values)
- Use dead reckoning: Estimate the robot's position and orientation based on its previous position and motion.
- Implement motion profiling(to reduce slip and jerk)
- Perform regular maintenance

Next steps:

- sensor fusion(kalman filter) and dead reckoning(if it's too complex, maybe not - teams have done without it) to reduce my odometry errors
- Make motion profile more reactive?(PID velocity, PID acceleration instead of motion profile(?))
- Test odom with turning and driving and then turning again

Name: Carol Wang

2023/05/09 @home

Programming log #6

At Tr/Tl = 6.5, varying 90.5-89.2 angle, varying too high and too low output

Overall too high output, so increase Tr/Tl: more turns for same distance

Tried multiple trials at 360 to calibrate

Compare to inertial -> changed to 6.47

Trials with:

100 velocity and 100 accel: around 24.4-5

200 velocity and 200 accel: around 24.5

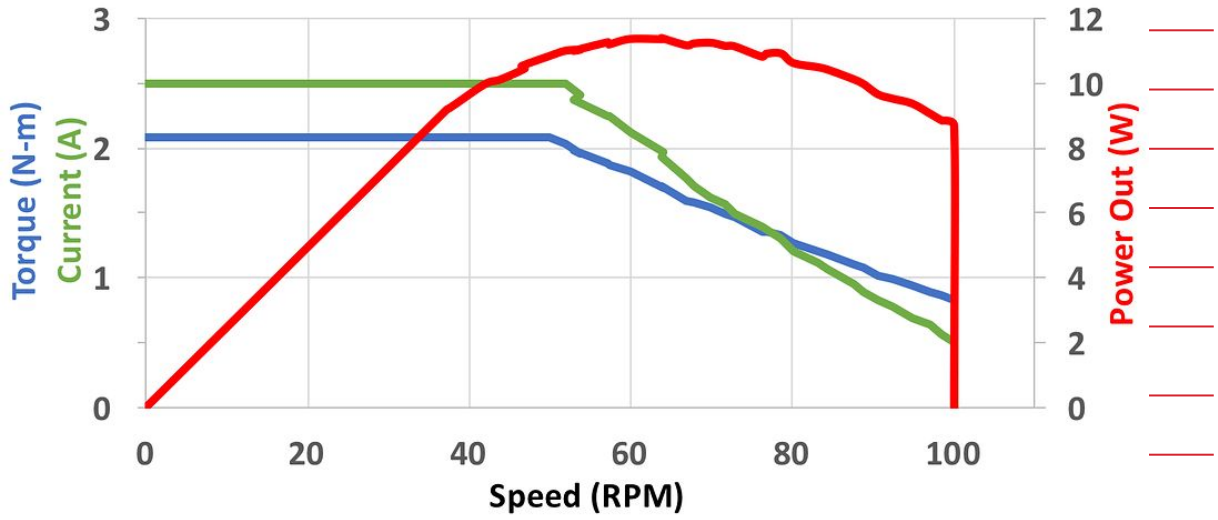
All around ¼ inch off

Vary 0 vs 0.8 gamma: no difference, around the same distance but gets a bit more error (around 0.1 inch) w/ 0 gamma-> consistently around 24.6 w/0.8 gamma-> consistently around 24.4

2023/05/12 @home

Programming log #7

Creating a team GitHub. Talked to main designer(Eva), decided to not use dead wheels because we need a drivetrain that can climb the PVC pipe.



Name: Carol Wang

2023/05/12 @home

Programming log #7 continued

Might consider writing a code for pushing where speed is just set to max torque, aka 50%

Next steps: finish odom testing/make turn and drive accurate, learn threading ->

PID/motion profiling thread at wait 100 ms, odometry thread at wait 10 ms, thread for cata/intake, thread for prepping??(like getting mechanisms ready while others are running)

Need to ask eva about strafing(can only track with horizontal dead wheel)

-> can attach dead wheels for skills only if possible

https://www.vexforum.com/t/motion-algorithm-comparisons-of-autonomous-routines-in-the-vex-robotics-competitions/79479

https://learnroadrunner.com/#frequently-asked-questions -> says ramsete better than pure pursuit??

https://www.reddit.com/r/FTC/comments/osnhna/odometry_or_not/

Comparing different autonomous control systems:

https://www.vexforum.com/t/motion-algorithm-comparisons-of-autonomous-routines-in-the-vex-robotics-competitions/79479/1

Name: Carol Wang

Strategies

Autonomous

At the start of the season, we brainstormed strategy to maximize both our autonomous period and skills scores.

Goal Get WP and win autonomous bonus. This way we raise our tournament ranking and increase our chance of winning elimination matches

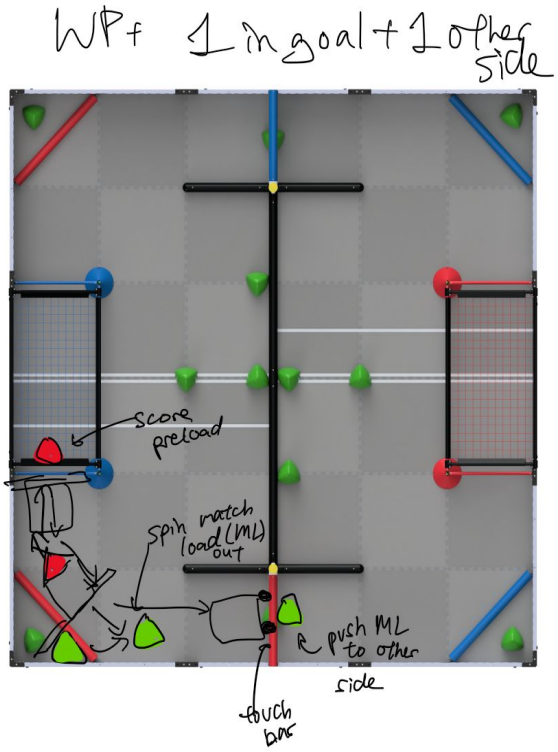
Requirements

- Maximize points scored
- Complete autonomous as quick as possible(only 15 seconds)
- Fulfil WP (Get triball out of matchload zone, touch elevation bar)
- Complete auton 8/10 times

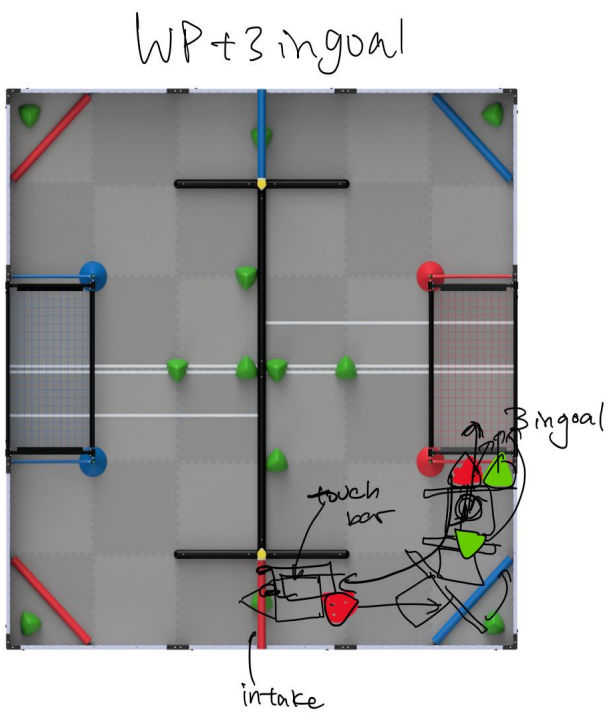
Final autonomous

- So maximize score by pushing triballs into goal
  - Triball under goal: 5pts
  - Triball on other side: 2pts
  - Hang: 0pts
- Score 6 triballs into alliance's goal
- Score each triballs in 2 seconds maximum
- Touch hang bar in 3 seconds maximum

Left



Right

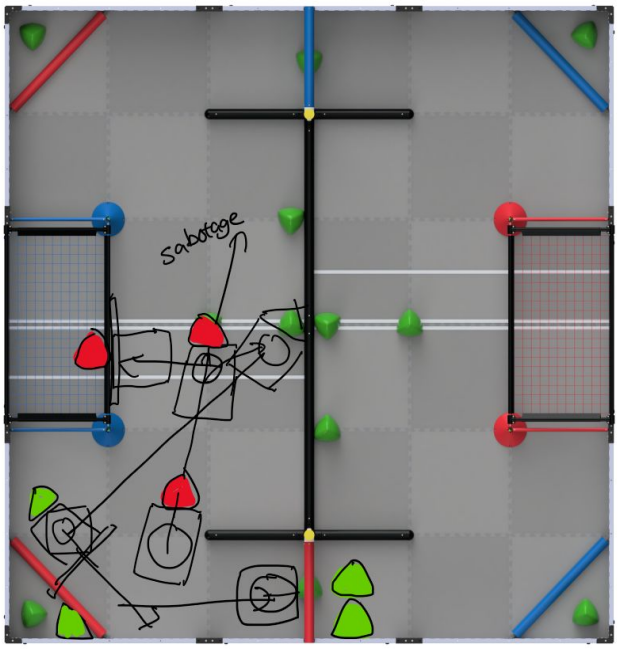


Inspiration from 392X: https://www.youtube.com/watch?v=fmbSN_4z8P8&ab_channel=392XSpeedZappers

Autonomous

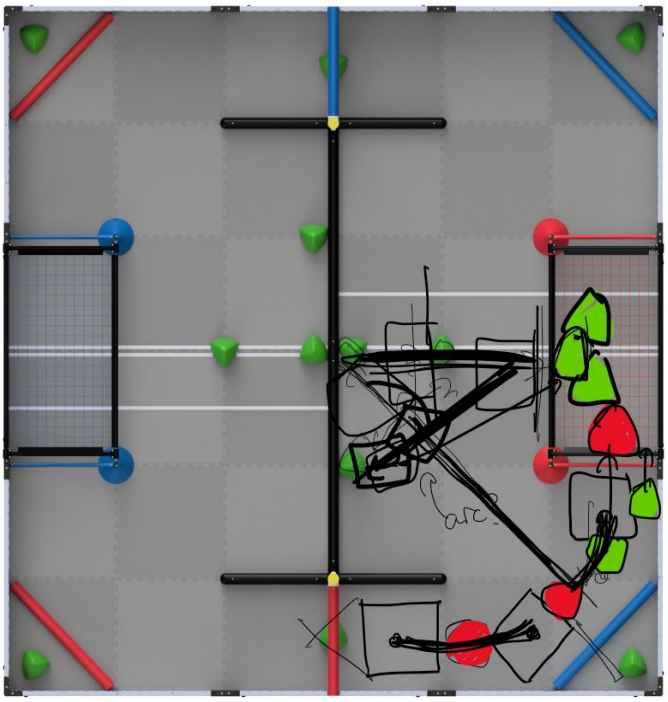
Left

1 PL in goal 2 green on other side + sabotage



Right

6 triball auton



6 triball inspiration from 66994A:
https://www.youtube.com/watch?v=l95d6lu0DLs&ab_channel=ROUNDABOUT

⭐Made separate WP and point-maximizing(PM) autons to increase reliability:

- WP are successful each time because they do not go into middle, where other teams can sabotage
- PM go into middle, because taking the risk is worth it for more points in elimination matches

Programming Skills

Goal Maximize programming skills score to

Requirements

- Complete in under 1 minute
- Use all 44 matchloads
- Complete all point-scoring operations: launching triballs, pushing, and hanging

Additional Considerations

- What control algorithms are most important?

Over Under's scoring mostly relies on the robot chassis' movement, since plowing triballs is such a big aspect. We focused on making the robot drive straight and have precise turns.

- Speed > precision

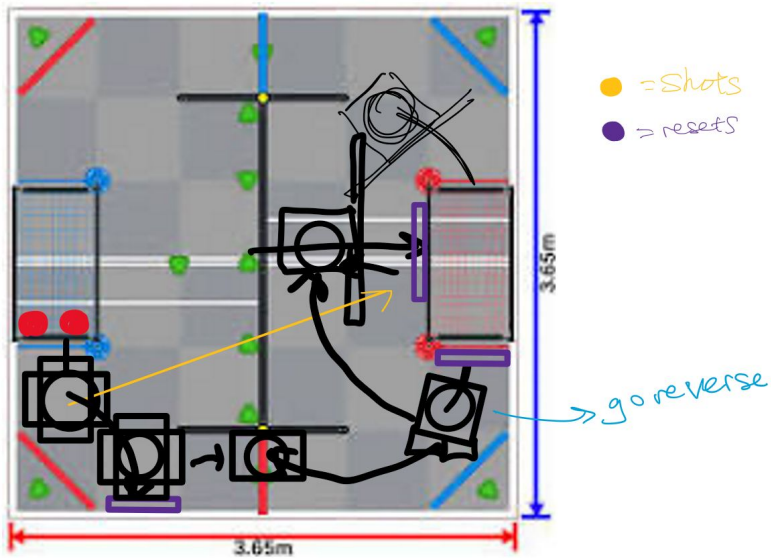
Triballs movement is hard to predict:
Speed and maximizing covered area > specific auton route

- Challenges:

Moving across barrier could mess up routine
Lots of the spaces are tight and one wrong move could result in the robot hitting a barrier

- Final goals:

Program quick plowing movements, but make moving around barriers/to hang bar very precise



- Robot resets at walls to keep recalibrating inertial sensor
- Robot curved sweeps are most efficient for pushing into triballs to sides
- Catapult triballs to front of net, sides are harder to push in

Points:
44 triballs over on other side
- 30 under net = 30 x 5 = 150 pts
- 14 on field = 14 x 2 = 28 pts
- Hang = 10 pts

Total = 188 pts

Logs

2023/09/11 @robotics lab

Programming log #8

- Mostly and introductory meeting for new members
- Changed wiring
- Attached radio

2023/09/12 @robotics lab

Programming log #9

- Made a GitHub: https://github.com/carolwangg/VEX-autonomous/tree/main
- Tested drive program(works somewhat-> physical problems with base hinder it)
- To do:
  - Attach pneumatics
  - Test drive program
  - Start autonomous(still need to come up with auton strat)
  - Start skills

  - Explain to Aleena:
  - Command based programming(done coding, not tested)
    - Separating code into robot commands, that can be executed consecutively or one after another
  - Odometry(not done coding, tested and does not work)
  - Motion profiling(done coding, tested and works)
  - Point to point(done coding, not tested)
  - Multihreading(not researched)

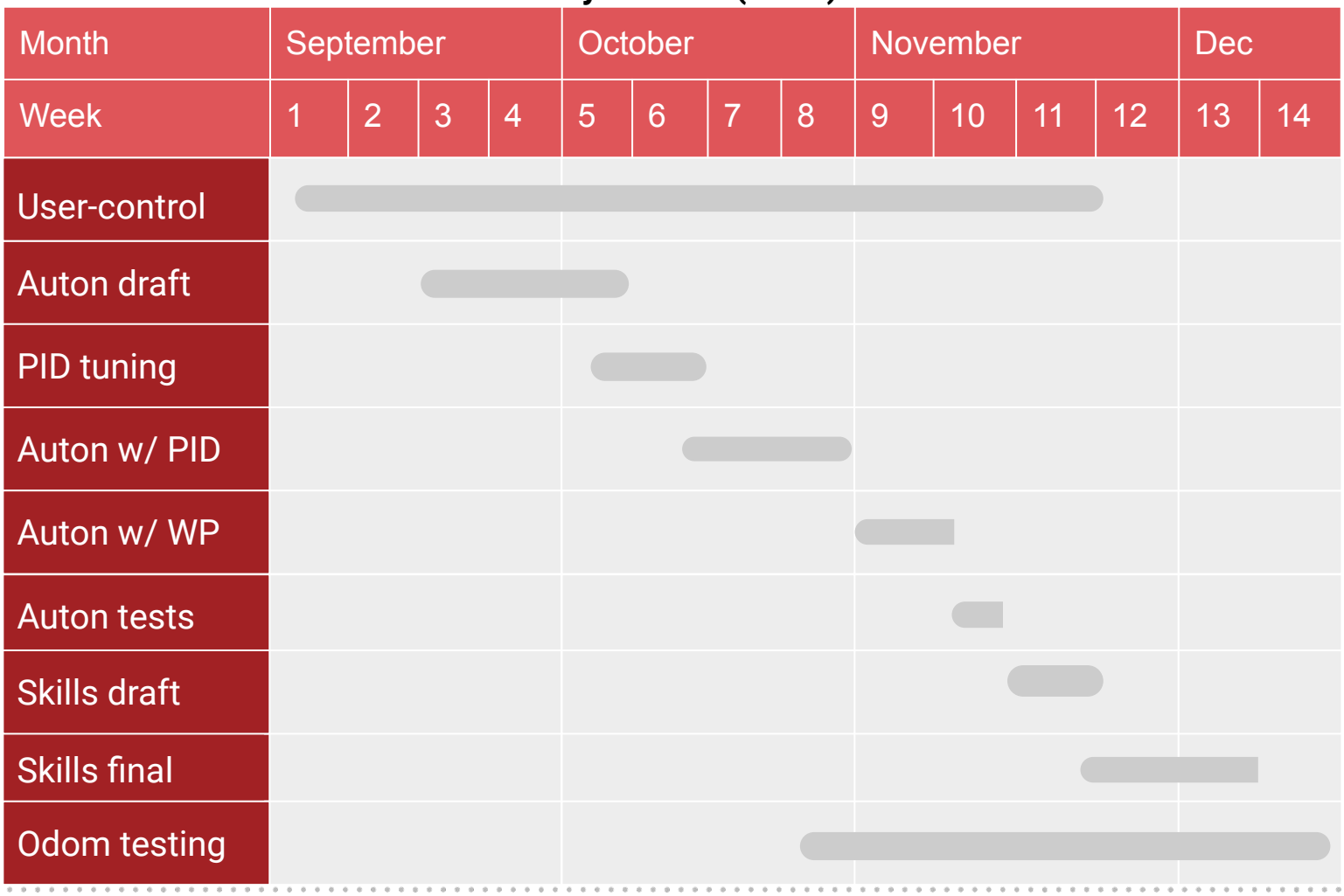
Name: Carol Wang

Gantt Chart

Goal To maximize efficiency and ensure all programs are ready **and tested** before the competition

Requirements Finish everything by a week before the Ottawa competition, which is on the second week of December.

Early-season (2023)



Terminology:
User-control: program for the joystick, for driver mode
Auton: autonomous period's program
Skills: programming skills program
Draft: a rough copy of a program, which will make the robot move to the right spots but inconsistently because no motion control algorithms are used
Final: a finished copy of a program, which includes PID, motion profiling, etc.
WP: Win Point

Programming log #10

https://www.vexforum.com/t/multithreading-in-python-vexcode-v5/103252

- Followed the code outlined in this forum post by jpearman to implement multithreading in odometry -> tested with different sleep() values and it worked
- Finished writing pseudo code for DriveTo(TURNS)/(MM)
  - Will be implemented using round robin to periodically update odometry, pid, and velocity each iteration of while loop
  - Can use functions for robot operations to simplify autonomous programming instead of using multiple while loops for each robot operation as before:
    - Ex:

```
while piddrivepower1(setpoint, frDrive.position(TURNS)) is not 0:
    driveforwardpercent(piddrivepower1(setpoint,frDrive.position(T
URNS)))
    wait(50, MSEC)
drivestop()
```

Turns into

```
driveto(setpoint)
```

To do:

- Check on Timed Run to see if any issues would arise while using in competition
- Tune drive PID(after building is completed) with Aleena
- Make sure all python functions work as intended
- Write pseudo code for TurnTo(DEGREES)
- Put finished code comparison in this Docs after finishing
- Calculate skills measurements

- After coding everything, decide whether to add ratio(36/84) to inside of functions
  - Instead of Motor turns -> wheel turns -> distance
  - Do motor turns -> distance
  - Already internalized for angle

Name: Carol Wang

Programming log #11

2775Josh — Zaypers — Dec '22

Yes, we are using algorithms!

The swerving movement is just full powering the motors, so like at the very beginning of the run when the robot swings off the roller to pick up the disc and hit the next roller, that's just motor voltages to make it nice and fast. And then the actual good algorithm is just a drive to point PID function that always tries to face the point it's driving to. No pure pursuit, no motion profiles.

2775Josh — Dec '22

We just have two pid loops going, a drive pid and a heading correction pid. The drive pid takes in as error the euclidean distance to the point, and the heading pid takes the error between the angle of the robot and the angle to the point (which we just get using atan2). Whenever the robot is within an x-inch radius of the desired point, it exits the pid loop and moves on. (Sometimes we use a radius of 2, sometimes 3.

Anomaly — Apr '19

Most everyone struggles with accurate base control in auton. At the highest level, 5225 developed a 3 tracking wheel solution that allowed them to set the world record in programming skills last year, and a handful of teams including 8059 and 139 have replicated their "odometry" based on a document they released last year. Team 5225 Introduction to Position Tracking Document 261

But at a less extreme level, for people just looking to take a step up from `move.relative` you may like the base control code I developed this year. Before I go any farther, I'd like to openly admit that I'm not a good coder in any capacity and my conventions and whatnot may be sloppy, but I nevertheless won 8 of my 11 autonomous periods at worlds and was overall very happy with my chassis control in auton. And when my robot did miss, it was entirely because I didn't spend enough time tuning my constants. The results I got on the competition field were perfectly replicable on all practice fields, regardless of antistatic.

The basic philosophy of my code is that the only possible reason an encoder value would not be representative of the true position of the robot is if the wheels slip or lift off the ground. As such, code that prevents the robot from accelerating or decelerating too fast and thus has no wheel slippage results in encoder values that are perfectly representative of the true position of the robot. To prevent too high an acceleration value, I used slew rate control when speeding up and a P loop when slowing down. 9605 and 169 use a similar concept (though I haven't seen their code) and PID with slew rate has been a concept forever, so I'm not trying to take credit for anyone else's work. But here is mine.

https://www.vexforum.com/t/very-simple-very-accurate-chassis-control-code-release/60819

- Might be worth making a pid drive + pid angle focus algorithm

For example:

Straight-line assist is common in most of the chassis PIDs I've seen made from more experienced programmers. If you look on github (okapi, tabor's 48, etc) you can find many people use it.

Its not too complicated - basically, you have one PID dedicated to the forward movement of the robot, and another one dedicated to keeping the robot straight.

With a generic PID implementation like I showed above, a simple loop might be:

```
PID distancePID(1, 0.1);
PID anglePID(0.01, 0.1);
double target = 100;

while(true) {
  //average of both sides
  double distance = (getLeftEnc() + getRightEnc()) / 2.0;
  //difference between left and right side
  double angle = getLeftEnc() - getRightEnc();
  //if left side is above right, angle will be positive,
  //which we will use to turn the robot more to the left

  double distancePower = distancePid.calculate(target, distance);
  double anglePower = anglePid.calculate(0, angle);

  double leftPower = distancePower + anglePower;
  double rightPower = distancePower - anglePower;

  //now you can set motor power
  delay(10);
}
```


Name: Carol Wang

2023/09/28 @robotics lab

Programming log #11 continued

- Things to look into:
  - Cheesy drive: https://wiki.purduesigbots.com/software/competition-specific/curvature-cheesy-drive
  - Okalib library: https://okapilib.github.io/OkapiLib/md_docs_tutorials_concepts_twodmotionprofiling.html
  - Road Runner: https://acme-robotics.gitbook.io/road-runner/tour/motion-profiling
  - JAR-template: https://github.com/JacksonAreaRobotics/JAR-Template/blob/main/include/autons.h
  - Slew rate
    - Minipid: https://github.com/tekdemo/minipid

2023/10/05 @robotics lab

Programming log #12

- Ace of base:
  - Finished program for Ace of Base competition, and tested
  - Everything works well, but robot is turning a little over target, especially after removing 0.5 second wait times after each turn
  - Expected reasons for inaccurate turns:
    - Because program was run on hardwood surface instead of regular foam mats(that Kp was tuned for)
    - Because all PIDs were tuned with robot moving from rest and stopping, not tuned for consecutive movements. Maybe lack of wait time between actions caused inaccurate movement
  - Other comments:
    - The pid is pretty slow, especially at the end part during stabilization
      - Only was to fix is allow more inaccuracy(which would be fine if odometry was used, because then each movement could be calculated from point to point instead of movement to movement)
        - Or to tune more
- To do:
  - Test Ace of Base on mats, with different rest times(to determine minimum *wait()* needed to still be accurate)
  - Work on odometry

Name: Carol Wang

2023/10/07 @ home

Programming log #13

After testing PID with multiple turns one after the other:

- Multiple turns, even without stops between them, don't increase error in angle

After testing PID with drive straight right after:

- Faster velocities while driving straight right after turning increases angle deviation away from zero by about 0.1 for every 10 RPM
- Longer wait times between finishing turning and driving straight seem to have little effect on minimizing deviation from zero
- ~~Conclusion: might need to conduct some trials to see how serious the error~~ is/whether there's a trend. If error is random(waiting longer or driving slower cannot minimize) might need to make drive straight PID

After testing odometry:

- Odometry has lowest error(within 0.2 inches) when pressing down hard on robot
- ~~Odometry has very high turning angle error(experimental-theoretical angle) and x and y value when turning. X value when driving straight is also pretty inaccurate~~
- Error size vs. turning amount/drive straight amount seems to be random
  - ~~Conclusion: I won't do anything about that for now, since error seems~~ decently small. I will retune turn/drive PID values later if it really bothers the autonomous run
- ~~Conclusion: need to get horizontal tracking wheel. Also need to test with motion profile to test if wheel slippage(which pressing down on robot eliminated) is avoidable or if programming needs to account for it.~~
- ~~To read:~~
  - https://gm0.org/en/latest/docs/software/concepts/odometry.html
- To do: lower 6.75 Tl/Tr value to 6.25(middle of wheel) if that doesn't make turning more accurate, minimize to 5.75
- ~~Once turning is accurate, test by moving robot straight, turn, sideways, stop(to test accuracy of x + y movements combined)~~
- If inaccurate, investigate code(particularly beta value, don't know where that ~~calculation even came from)~~
- If still inaccurate, start researching dead wheels. Finish by monday

Name: Carol Wang

2023/10/08 @ home

Programming log #14

Checked autonomous theory against this video:
https://www.youtube.com/watch?v=_T6KHywSP58&ab_channel=Queen%27sVEXURoboticsTeam

- Found it to be mostly correct, but to make odom a little more accurate, assumes robot drives in arcs with no strafe(instead of assuming driving perfectly straight with no strafe),
- Changed to algorithms that are a little more complex

Before:

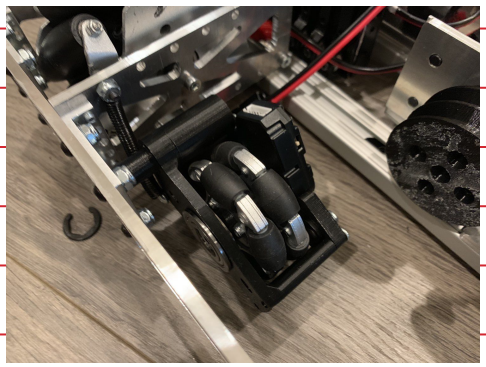
```
beta = theta + (theta_change/2)
x += distance_center*math.sin(beta)
y += distance_center*math.cos(beta)
theta += theta_change
```

After:

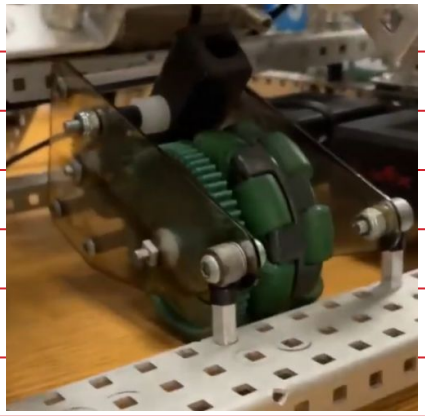
```
if(theta_change == 0):
    dy = distance_center
else:
    dy = 2*((distance_right/theta_change)+Tr)*(math.sin(theta_change/2))
beta = theta + (theta_change/2)
x += dy*math.sin(beta)
y += dy*math.cos(beta)
theta += theta_change
```

By putting my python program into Programiz(online IDE) and testing it by directly typing in motor.turn values, I was able to test whether the math behind the program was accurate or not, and it was. All values came out as expected, tested against me doing the math by hand.

Researched odom pods
Odometry pods:



Reference: FTC Team 8802



Vex team 7983S

Name: Carol Wang

Programming log #15

- Tested Ace of Base program on competition mats
  - Worked well, was consistent: robot arrived at same endpoint on mat each time, and error on 90 degree turns was negligible
- Retuned PIDs
  - Retuned PIDs to decrease settling time of PID, thus making the autonomous go faster. This decreases accuracy, but after testing, the PIDs still seem to be accurate enough to consistently reach the same endpoint.

Before:

TURN PID

```
if 0<turnpower<1:
      turnpower=1
elif 0>turnpower>-1:
      turnpower=-1
if -0.5<error<0.5:
      turnpower = 0
```

DRIVE PID

```
if 0 <=drivepower < 2:
      drivepower = 2
if -2 <drivepower <0:
      drivepower = -2
if -0.01<error<0.01:
      drivepower=0
```

After:

TURN PID

```
if 0<turnpower<3:
      turnpower=3
elif 0>turnpower>-3:
      turnpower=-3
if -1<error<1:
      turnpower = 0
```

DRIVE PID

```
if 0 <=drivepower < 8:
      drivepower = 8
if -8 <drivepower <0:
      drivepower = -8
if -0.1<error<0.1:
      drivepower=0
```

- Built one test-purpose odometry pods out of aluminum(the real competition ones will use plastic), 2.75” wheels and old VEX wheel encoders(red)
  - To get all the spacing right before plastic and other competition-use pieces are cut
- Next time:
  - Finish second odom pod, attach pods to chassis
- To read: https://www.vexforum.com/t/7842f-2018-19-robot-showcase-flywheel-control-odometry-engineering-journals-more/65170 (for auton ideas, from an old 2018/19 team)

Name: Carol Wang and Aleena Long

2023/10/13 @ robotics lab

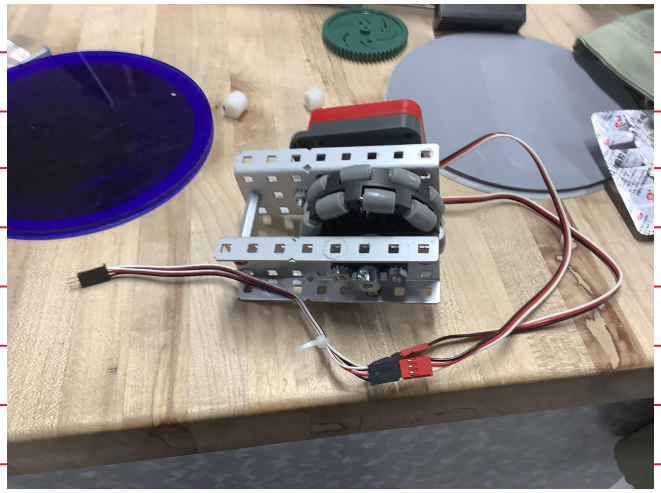
Programming log #16

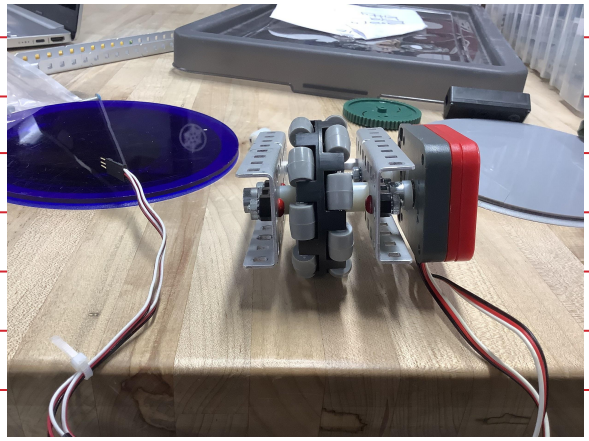
- Finished auton pods and tried attaching them to chassis to test
- They are too bulky, so next time we will make new auton pods with plastic that will fit into the chassis better

2023/10/16 @ robotics lab

Programming log #17

- Drafted out the design for the plastic side pieces of the odom pod
- Started cutting plastic
- Next time:
  - Finish odom pods and finish coding programming skills run





Name: Carol Wang and Aleena Long

Programming log #18

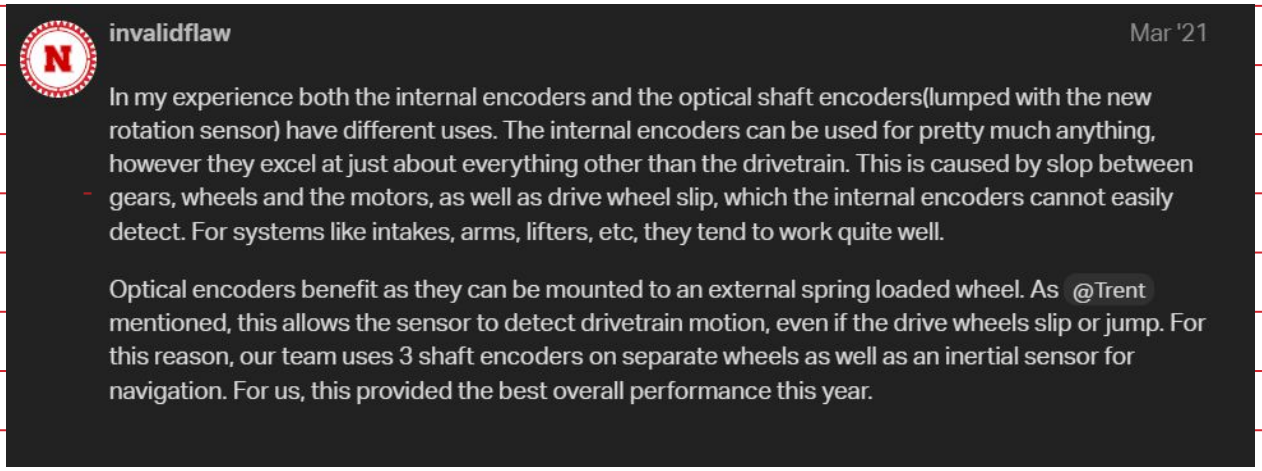
- Widened hole so screw for encoder can go on straight
- Found that screwing encoder into plastic tightly is the main reason that shaft spins with more friction

Next time:

- Screw shaft onto chassis, maybe secure encoder onto chassis too instead of onto plastic sides of odom pod
- Program velocity ramp up using motion profiles

To read:

- https://www.vexforum.com/t/tracking-wheels-for-odometry-using-the-potentiometer-v2/92369
- https://www.vexforum.com/t/optical-encoders-vs-v5-motor-encoders/89332/6



invalidflaw Mar '21

In my experience both the internal encoders and the optical shaft encoders(lumped with the new rotation sensor) have different uses. The internal encoders can be used for pretty much anything, however they excel at just about everything other than the drivetrain. This is caused by slop between gears, wheels and the motors, as well as drive wheel slip, which the internal encoders cannot easily detect. For systems like intakes, arms, lifters, etc, they tend to work quite well.

Optical encoders benefit as they can be mounted to an external spring loaded wheel. As @Trent mentioned, this allows the sensor to detect drivetrain motion, even if the drive wheels slip or jump. For this reason, our team uses 3 shaft encoders on separate wheels as well as an inertial sensor for navigation. For us, this provided the best overall performance this year.

Name: Carol Wang and Aleena Long

2023/10/30 @robotics lab

Programming log #21

- Aleena finished half of her left side auton, I finished a full right side auton
- Made a finished odom pod
- To do(= to do tomorrow, since chassis will be taken by builders):
  - Make full auton (that gets solo win point)
  - Make a ramp up function(probably will just use motion profiling) to test odom with pods
  - More auton combos - try to get odom done FAST so i can experiment with pathfinding?
  - Vision sensor program to auto get balls??
    - Game elements this year are sparse
    - I have nothing to do when Kitty is practicing driving
- To do in distance future:
  - Pre-auton visual selection✅
  - Vex build-your-autonomous

2023/11/16 @robotics lab

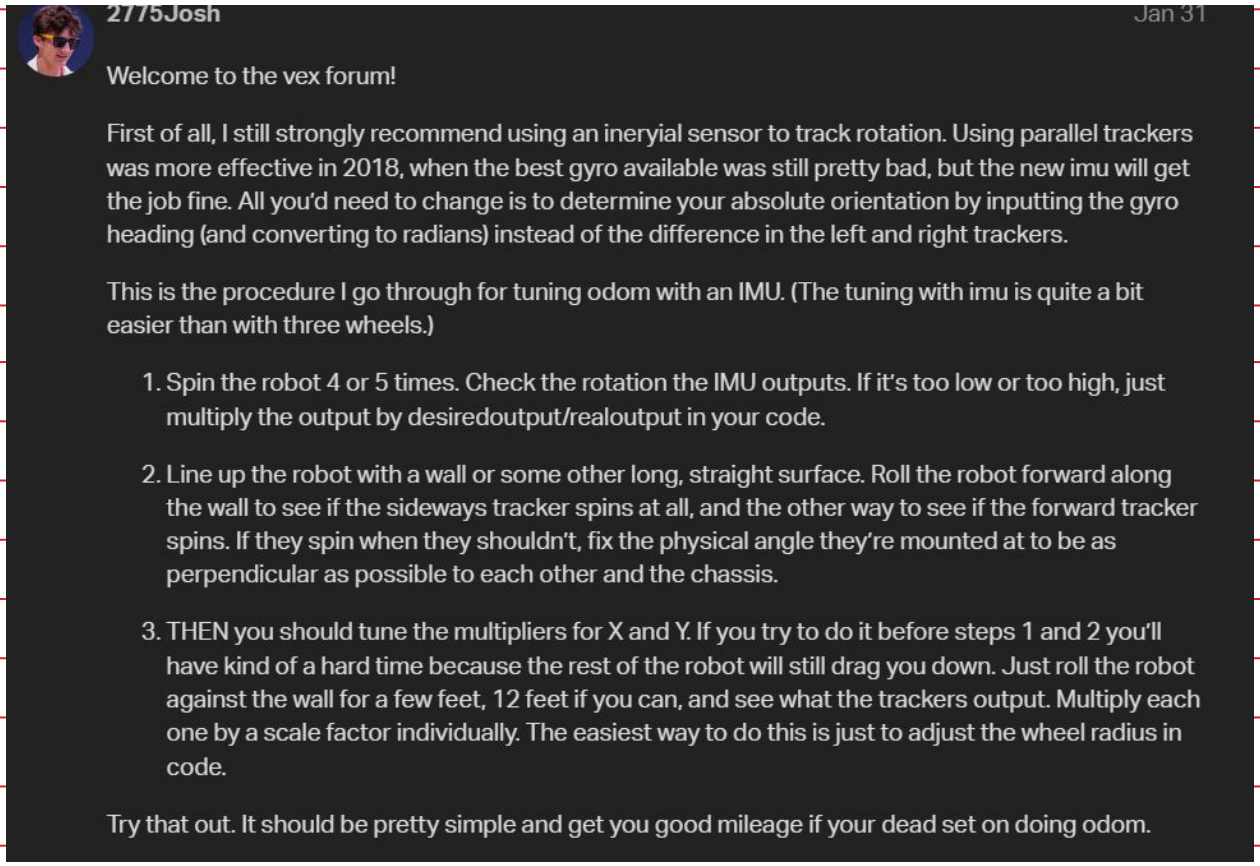
Programming log #22

- Tried to attach odometry pod to chassis via standoff and collar, did not work
  - The standoff was too short/hung too high of the ground
    - Can't make it longer since the screw used to attach it was the 2" black screw, which is the longest possible
    - Need to use headless screw (no more in stock)

Name: Carol Wang and Aleena Long

Programming log #23

- Recalibrated all odometry(Tr, Tl, Ts, gear ratio, diameter) and some motion profiling variables to new chassis
  - Calibrated odometry wheel diameter by trial and error, tweaking value until calculated angle was close to inertial sensor provided angle
    - Ran several tests at 90 degrees, still off by 0.2~0.7 each time not ideal
    - May need to just use inertial sensor in odom
  - Out of three odometry tested, normal one seemed to work best(versus odom with inertial sensor instead of calculating from wheel turns, and odom with horizontal tracking wheel factored in)
- Drive to point: https://www.vexforum.com/t/doubts-about-odometry/100560/4
- Odometry flowchart: https://www.vexforum.com/t/flowchart-for-learning-programming-particularly-odometry/95479/3
- Increase Tr/Tl
- Use IMU for turning NOT wheels (according to 2775Josh sensor is better/more accurate)



2775Josh Jan 31

Welcome to the vex forum!

First of all, I still strongly recommend using an ineryial sensor to track rotation. Using parallel trackers was more effective in 2018, when the best gyro available was still pretty bad, but the new imu will get the job fine. All you'd need to change is to determine your absolute orientation by inputting the gyro heading (and converting to radians) instead of the difference in the left and right trackers.

This is the procedure I go through for tuning odom with an IMU. (The tuning with imu is quite a bit easier than with three wheels.)

1. Spin the robot 4 or 5 times. Check the rotation the IMU outputs. If it's too low or too high, just multiply the output by desiredoutput/realoutput in your code.
2. Line up the robot with a wall or some other long, straight surface. Roll the robot forward along the wall to see if the sideways tracker spins at all, and the other way to see if the forward tracker spins. If they spin when they shouldn't, fix the physical angle they're mounted at to be as perpendicular as possible to each other and the chassis.
3. THEN you should tune the multipliers for X and Y. If you try to do it before steps 1 and 2 you'll have kind of a hard time because the rest of the robot will still drag you down. Just roll the robot against the wall for a few feet, 12 feet if you can, and see what the trackers output. Multiply each one by a scale factor individually. The easiest way to do this is just to adjust the wheel radius in code.

Try that out. It should be pretty simple and get you good mileage if your dead set on doing odom.

Good link for new teams: https://www.vexforum.com/t/new-team-guide-to-vrc/92764/2

Name: Carol Wang

2023/11/26 @robotics lab

Programming log #24

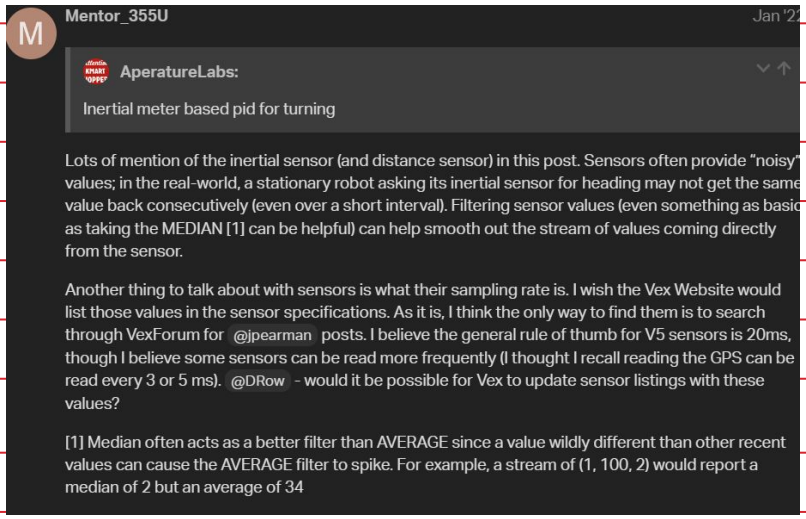
- Tuned wheel diameters and made horizontal into vertical encoder wheel
  - Couldn't figure out if drive wheels or encoder was more accurate, it seemed pretty random sometimes some were right, sometimes the other was
- Used inertial sensor for turning
  - After tuning the drive wheels' diameter, turn was ALWAYS more to the right than needed, for some reason -> for 180 degree turn, would output 210 degrees
- Tested by driving around and looking at numbers
  - Y is within ~1 inch error, x is very big error(wheel encoder always too much x, drive wheels always too little x)
- To do next:
  - Add the horizontal encoder wheel to see if x translation becomes more accurate
    - When I tested earlier with only horizontal and no vertical encoder, y was always smaller than real value and x always too big
    - Probably just need to lower diameter of horizontal encoder wheel, so has less effect on results

2023/11/28 @robotics lab

Programming log #25

- Finished the right side autonomous
  - One gets three triballs in
  - One gets two triballs and touches pole
- Left side autonomous almost done
  - Gets triball out, but unreliably
- Start programming skills
  - Shoots 40 balls into goal
- To do:
  - Make the PIDs quicker by making settling velocity quicker
  - Add a ramp up function
  - Right side autonomous sometimes bumps into side of goal(makes it unreliable), fix that
  - Potentiometers track absolute position, attach to arm if we are using it long term
  - Average out encoder + inertial values for each turn

Name: Carol Wang and Aleena Long



Mentor_355U Jan '2

ApperatureLabs:

Inertial meter based pid for turning

Lots of mention of the inertial sensor (and distance sensor) in this post. Sensors often provide "noisy" values; in the real-world, a stationary robot asking its inertial sensor for heading may not get the same value back consecutively (even over a short interval). Filtering sensor values (even something as basic as taking the MEDIAN [1] can be helpful) can help smooth out the stream of values coming directly from the sensor.

Another thing to talk about with sensors is what their sampling rate is. I wish the Vex Website would list those values in the sensor specifications. As it is, I think the only way to find them is to search through VexForum for @jpearman posts. I believe the general rule of thumb for V5 sensors is 20ms, though I believe some sensors can be read more frequently (I thought I recall reading the GPS can be read every 3 or 5 ms). @DRow - would it be possible for Vex to update sensor listings with these values?

[1] Median often acts as a better filter than AVERAGE since a value wildly different than other recent values can cause the AVERAGE filter to spike. For example, a stream of (1, 100, 2) would report a median of 2 but an average of 34

- ~~Implement function to check for stopping once odom is done:~~

Run into something, or need to know if you are playing an auton tug of war? Use the inertia.collision or motor.current or velocity functions to know if you are going the speed you should be going.

Picking up an object? Use the vision sensor to write a drive straight PID, replacing your inertial meter with the horizontal object position of your vision sensor, so that the colored object stays centered in the camera while you drive towards it.
...Then use a line reader or distance sensor to make sure you grabbed the object. Pretty easy coding here.

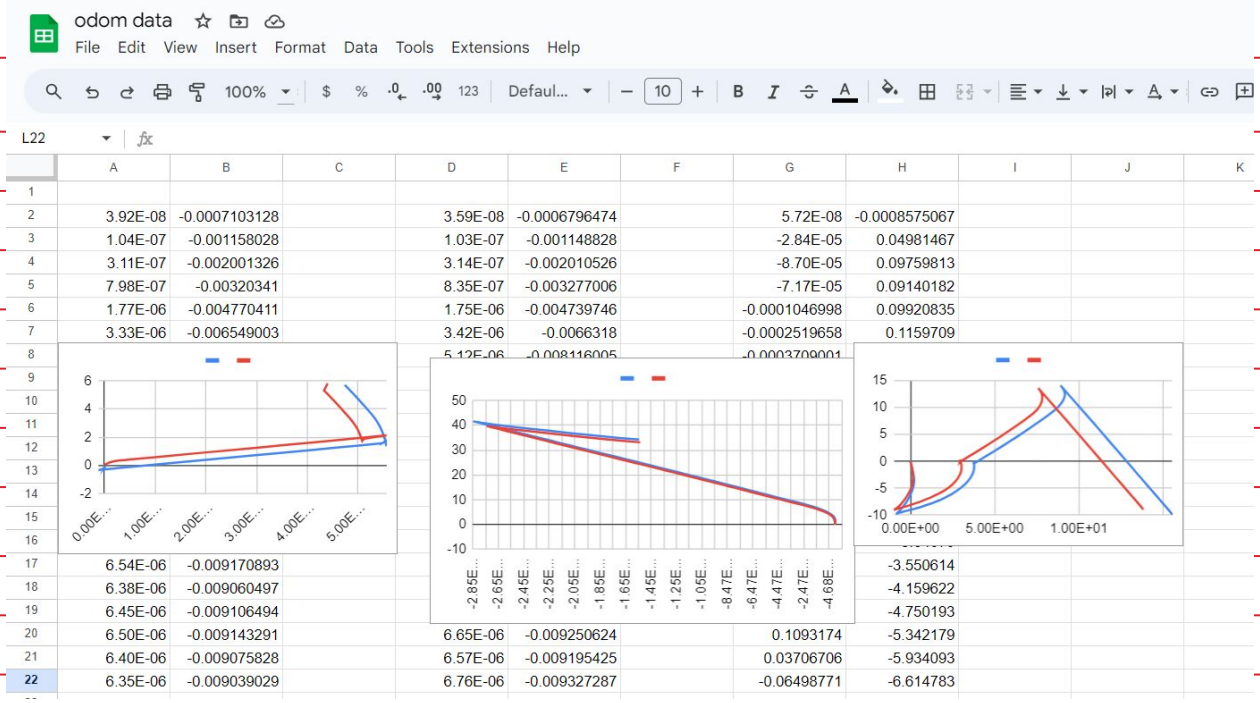
Furthermore, if objects are placed well, you can use trig+inertial meter when you grab an object to calculate your position.

- To read: https://theol0403.github.io/7842B-Journal/ control algorithm documentation from Theo Lamay
  - Motion profiling: https://lucky-bush-4c9b.theol0403.workers.dev/
    - Introducing the idea that better movement = more consistent movement and not more reactive movement
- ~~https://www.vexforum.com/t/list-of-all-vexcode/83052/6~~
- Fast auton coding: https://www.vexforum.com/t/tutorial-how-to-code-a-vex-autonomous-in-less-than-5-minutes/69205
- https://docs.wpilib.org/en/stable/docs/software/advanced-controls/controllers/profiled-pidcontroller.html

Name: Carol Wang and Aleena Long

Programming log #26

- Odometry data
- Red is tracking wheel, blue is chassis wheels



	A	B	C	D	E	F	G	H	I	J	K
1											
2	3.92E-08	-0.0007103128		3.59E-08	-0.0006796474		5.72E-08	-0.0008575067			
3	1.04E-07	-0.001158028		1.03E-07	-0.001148828		-2.84E-05	0.04981467			
4	3.11E-07	-0.002001326		3.14E-07	-0.002010526		-8.70E-05	0.09759813			
5	7.98E-07	-0.00320341		8.35E-07	-0.003277006		-7.17E-05	0.09140182			
6	1.77E-06	-0.004770411		1.75E-06	-0.004739746		-0.0001046998	0.09920835			
7	3.33E-06	-0.006549003		3.42E-06	-0.0066318		-0.0002519658	0.1159709			
8											
9											
10											
11											
12											
13											
14											
15											
16											
17	6.54E-06	-0.009170893						-3.550614			
18	6.38E-06	-0.009060497						-4.159622			
19	6.45E-06	-0.009106494						-4.750193			
20	6.50E-06	-0.009143291		6.65E-06	-0.009250624		0.1093174	-5.342179			
21	6.40E-06	-0.009075828		6.57E-06	-0.009195425		0.03706706	-5.934093			
22	6.35E-06	-0.009039029		6.76E-06	-0.009327287		-0.06498771	-6.614783			

To do:

- Write drive to distance function
- Right side auton: Get only match load and closest triball in auton
- Clean up and comment all code to show at club
- Figure out arcs motion profiling
- Code preauton w controller and text ✅
  - Barebones, just printing auton name in middle of screen, and buzzing 3 times when auton selected(a pressed), as well as refresh sensor values (set all motor values to zero, calibrate sensor) when R2 pressed
    - paste(“calibrated!” on controller screen when calibrated)
    - Three buzzes after chosen, “Auton#: , Calibrated:Y/N” for chosen and calibrated
- Make it easier to get values for auton ✅
  - Make a program just for that, put in slot 8
  - L1 to refresh, run at max 20%, X to print avgturns() and inertial.heading()
- When I finish motion profiling calibration/know how many turns/inches for robot, I can calculate values for everything at home
    - Draw out lines and arcs on Desmos mapping of field
- Drive code:
  - Figure out how to print temp. values on controller

Name: Carol Wang

2023/12/14 @robotics lab

Programming log #27

- Finished coding programming skills
  - Very inconsistent near the end, could have used distance sensor to calibrate robot each time it drives towards a wall instead of using wheel turns alone
  - Resets heading near the beginning and the middle, so inertial_heading() is pretty accurate
- Finished pre autonomous auton selector
  - Put in a preAuton variable that checks if program is in pre_autonomous() mode, if so, auton selector runs
- To do:
  - Write PID portion of programming notebook
    - file:///C:/Users/czjwa/Downloads/515Revision_Notebook.4.29.2023.pdf
  - Implement distance sensor to check for wall after long drive??(in skills run - can even use when I alr have odom to recalibrate odom every few steps)

2023/12/18 @robotics lab

Programming log #28

- Implemented the Drive-Copy code for autonomous - it periodically (every 35 msec) checks robots velocities while the driver is driving, and prints them out so robot can follow those velocities at the same time stamps - creates a clone of driver's run
  - Did not work because printing to console results in spelling issues at high speeds
    - Like "frset_velocity(33.44.3, PER)" instead of "frDrive.set_velocity(33.44, PERCENT)"
    - Tried to fix by storing in array and printing all after -> still has errors in printing, and for some reason it ends up repeatedly printing the same values three times
    - To debug: just print while the driver is driving, but only print integers and commas to reduce printing volume/room for error

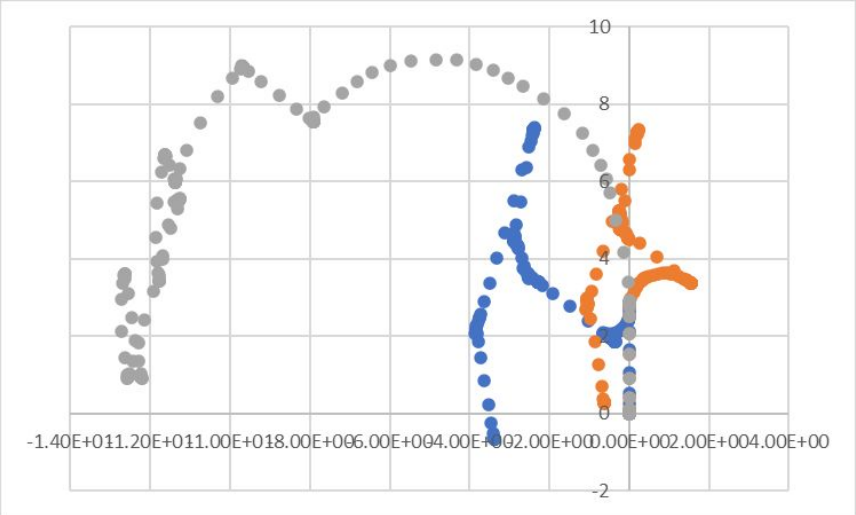
Winter Break until Jan. 9th

Name: Carol Wang and Aleena Long

2024/01/07 @home

Programming log #29

- Attached both odom wheels
  - Calibrated their wheel diameters to ~0.2 off for 30 inch drives-> still need to calibrate more
  - Straight one is slightly wobbly because no space to reach it to tighten more
  - Still need to elastic them to ground
- Tested odom
  - Weird results with algorithm that contains perpendicular back wheel
    - Need some way to diminish impact of back wheel, it is +/- ~0.4 to y calculation as well, not just x…
- Grey is calc with one left and one back odom wheel, blue is calc with two drive wheels, orange is calc with one left odom wheel



- One odom is normally not better than two…but in this case it seems more accurate
  - Difference between blue and orange is likely because I had not finished tuning yet at that point
- Still need to test turns though, seems as if algorithms break down most during curves/turns
- **To do:**
  - Test print algorithm with different turns (45, 90, 180 degrees)
  - Test curved driving (fix one side, move the other, use that to calibrate sTl(left odom wheel difference from center)
  - Also calibrate horizontal back wheel with curves
  - Test motion profiling with calibrated wheel circumferences
  - Read: https://file.tavsys.net/control/controls-engineering-in-frc.pdf

Name: Carol Wang

2024/01/08 @home

Programming log #30

- Calibrated wheel size for motion profiling
  - Weird thing where the farther the distance, the smaller wheel_circumference needs to be…so optimal wheel circumference differs for 12 vs 24 vs 48 vs 72 inches
  - - is it because of error in time? Causing longer trials to go on shorter or longer than they should….
- Calibrated left odom wheel distance from center
  - Moved the robot in arc with hand, if the length shown is too short, wheel distance from center needs to be smaller, if too long, increase distance from center
- Tested odom movement vs pid
  - PID more precise
  - Odom more accurate(get's closer to actual (x, y) goal) but less precise

View following photo:

- Tweaked odom alg to have both follow angle and distance pid overlapped
  - Causes a lot of variation in resulting position, but resulting position is way closer to target position(compared to PID)

2024/01/09 @robotics lab

Programming log #31

- Tested odometry on VEX field
  - Horizontal odom wheel left marks on the VEX tiles…
  - Vertical odom wheel recorded 2x larger length than real length…
    - Math accidentally changed/ wrong now??
  - Motor wheels were more accurate than vertical wheel…
- Tested VEX programming skills run
  - No longer working because driving is curving left, because chassis getting old more friction
  - Motors no longer as strong
  - More weight on right side of chassis- possible cause of issue
    - Try to put some weight(as much as the controller) on the left side to counterbalance

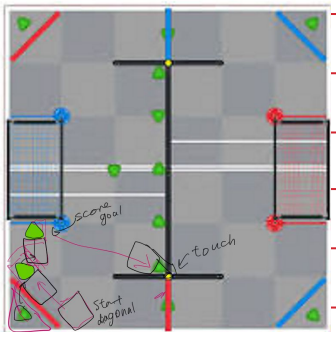
Name: Carol Wang

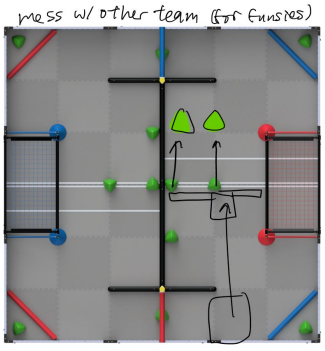
Programming log #33

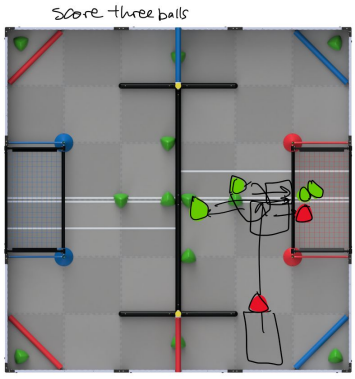
- Attached inertial sensor by two points on chassis, hopefully more straight and secure
- Started tuning PIDs - finished drive PID, was the same tuning which makes sense since max wheel speed and wheel circumference did not change
- Started tuning turn PID - the previous minimum 5 PERCENT velocity deadzone is too big. Need to adjust that.
- To do (by Monday):
  - Finish turning PID
  - Finish planning autons
  - Finish ramp up algorithm with motion profiling
  - Have Aleena tune the turning PID and explain to her how the motion profiling works
  - Wiring with Aleena and Delilah
  - Finished tuning turn PID
  - Finished most of custom wiring

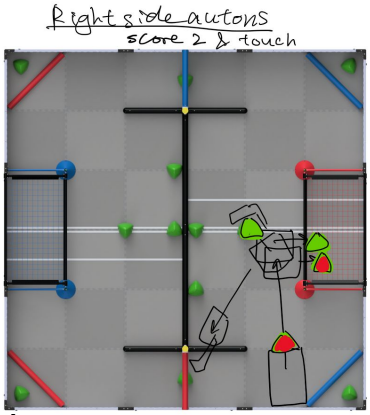
Redoing auton:

- In elimination match at Sudbury, other team had 3 triball auton as well-> need to increase our max triballs to continue getting auton bonus
- Observed online that a lot of teams drive into neutral zone during autonomous to mess up other team's auton
  - For WP auton do not interact with triballs in middle, so it does not get messed up by other team









Name: Carol Wang, Aleena Long, Delilah Hao

2024/01/30 @robotics lab

Programming log #34

- Finished custom wiring and pneumatic wiring
- Started part of right side auton -> now gets match load out and pushes preload into goal

2024/02/02 @robotics lab

Programming log #35

- User control:
  - ~~Programmed cata down command~~
  - Programmed hang
  - Both work now!
- ~~Autonomous:~~
  - Finished 2 triballs part for right auton
  - Consistently gets second one in, other one rolls away before intaking ½ the time
    - Bc intake has no back/fwd margin of error, so often touches triball before intaking, sending triball rolling ~~unpredictably~~
    - Second ball either gets intaked perfectly, or hits intake then rolls to bar, so it stops and gets intake well
    - ~~Movement PID pretty precise~~

- To do:
  - ~~Finish left side autonomous for WP~~
  - Finish right side 6 triball auton

Name: Carol Wang and Aleena Long

2024/02/05 @robotics lab

Programming log #36

- Finished left WP
  - Kinda jerky and slow/not smooth - distances travelled are too short to go fast, so can I somehow mix them?
- Tested curved turns on test bot
  - At constant velocity(50%), it works
- Next:
  - Finish right WP
  - Figure out how to follow a curving path?
  - Finish pre_autonomous(bring brain home if possible)
  - Tune cata positioning at beginning of programming skills(was stopping cata bc it was right against the wall) and make one last curved left push at end + elevation
  - Make sure all pneumatics are not leaking

2024/02/06 @robotics lab

Programming log #37

- Kept calibrating right side 6 triball auton
- Tested programming skills
  - Need to make the robot angle shoot towards center more -> right now its mostly right side
  - Need to change the distance while driving to other side to be longer, so robot goes right against bar/side wall and sweeps triballs more effectively

Name: Carol Wang and Aleena Long

Theory/Equations

PID

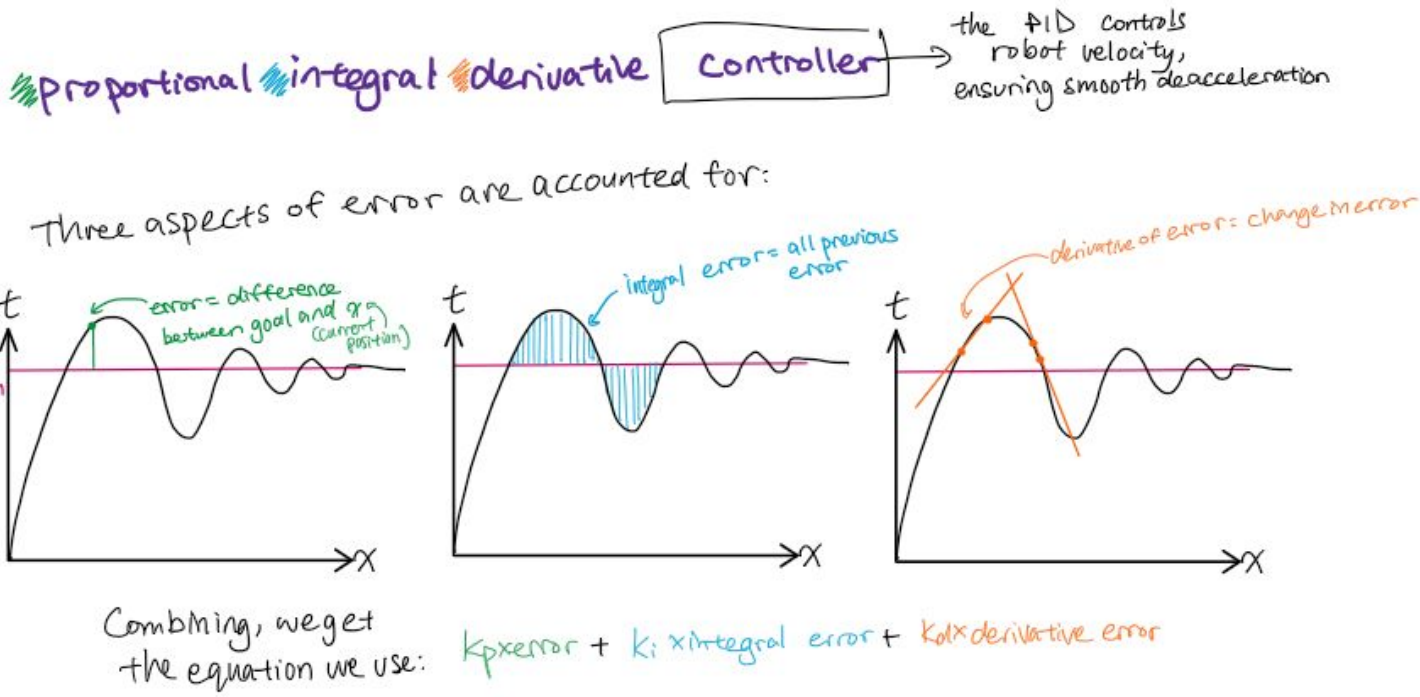
Goal

We need to move the robot to a target distance or angle, in the quickest and most accurate way possible.

Requirements

- Must work for both turning and driving straight
- Must raise accuracy to 8/10

Explanation



Final Code

```
902  def piddrive1(setpoint, curposition):
903      Kp = 39 #tuning notes: 38 for smooth stop
904      error = setpoint - curposition
905      drivepower = error*Kp
906      if drivepower> 100 or drivepower < -100:
907          #makes sure drivepower not over 100%
908          if drivepower > 0:
909              coeff = 1
910          else:
911              coeff = -1
912          drivepower = 100*coeff
913      if 0 <=drivepower < 9:
914          #make sure robot doesn't ever go unde
915          drivepower = 9
916      if -9 <drivepower <0:
917          drivepower = -9
918      if -0.05<error<0.05:
919          drivepower=0
920      return drivepower
```

```
844  def pidturn1(setpoint, curposition, start):
845      #PID for turning(to a point)
846      #if it turns over, will correct itself
847      #when setpoint > curposition, will turn r
848      integer = abs(setpoint - start)
849      Kp = 1/integer*9
850      Ki = 0
851      Kd = 0
852      global integralturn
853      global lasterrorturn
854      error = setpoint - curposition
855      # integralturn += error
856      # derivative = error-lasterrorturn
857      # lasterrorturn = error
858      #2.8 is the lowest possible voltage to st
859      turnpower = error*Kp#+integralturn*Ki+der
860      if 0 <=turnpower < 3:
861          turnpower = 3
862      if -3 <turnpower<0:
863          turnpower = -3
864      return turnpower
```

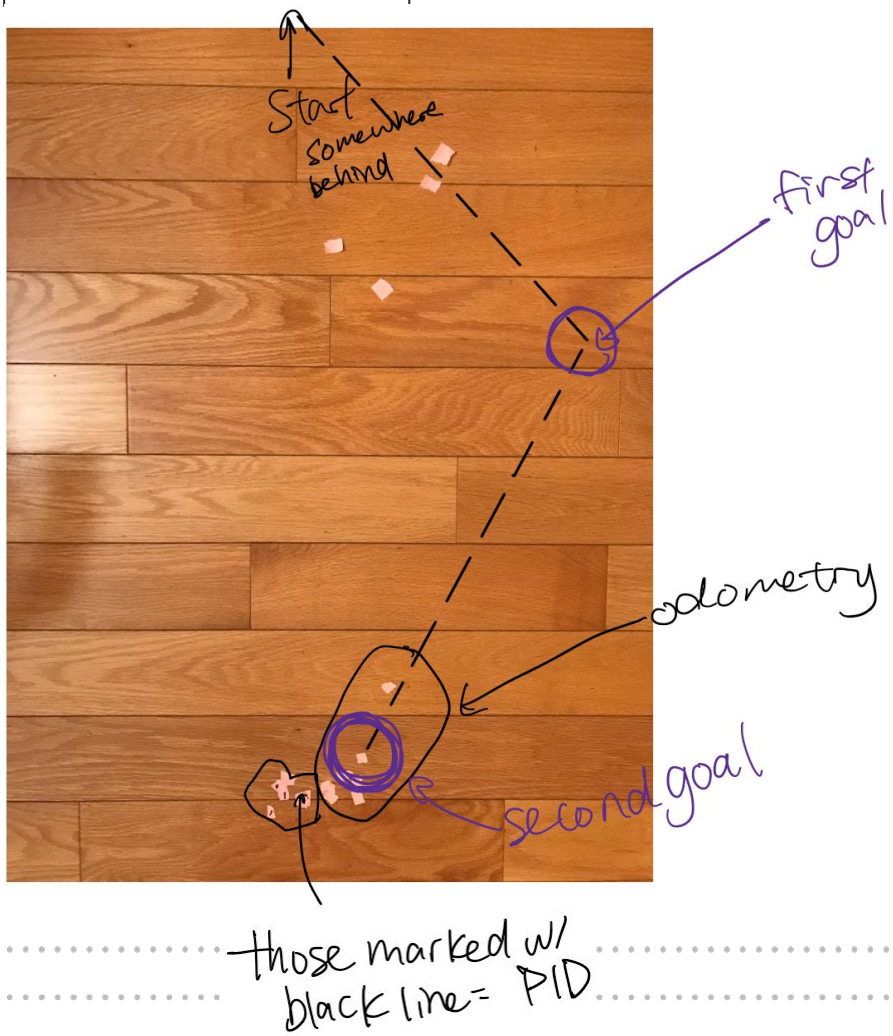
PID Tests

Goal Make sure the PID is working properly, especially at longer driving/turning lengths

Requirements

- Conduct multiple trials at the same conditions
- Drive for more than 24 inches

Results



Observations

- ★ Five trials were completed
- ★ High precision: Each trial average 0.5 inch off from each other
  - Two were right on top of each other
- ★ PID is effective

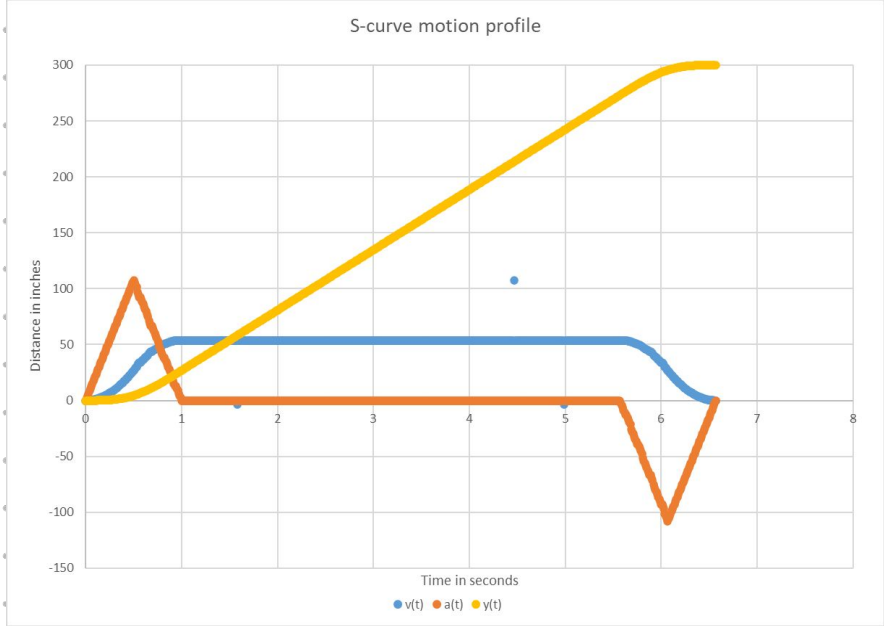
Motion Profiling

Goal We want to move the robot smoothly across planned routes. We want to minimize jerk to protect motors', and we want to move in curved paths for more mobility.

Requirements

- Minimize jerk and harm to motors from rapid acceleration
- Increase autonomous consistency to 9/10
- HOWEVER do not decrease speed of movement/sacrifice speed for precision

Explanation



🔷KEY:
- Orange = acceleration
- Blue = velocity
- Yellow = position

The equation we use to calculate our motion profile is based off this youtube video: https://www.youtube.com/watch?v=IpCWwYqE36Y&t=37s&ab_channel=DamonSisk Which is based of the paper "sinusoidal Velocity Profiles for Motion Control" by V. Arevelo

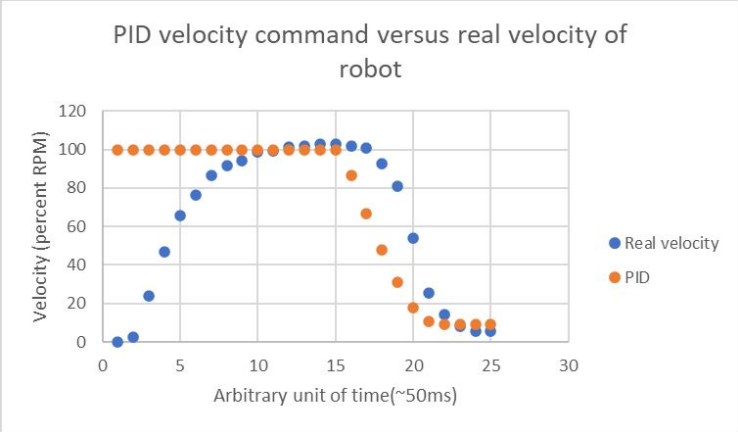
As shown above, it is a piecewise function that uses time(on the x-axis) to calculate velocity (on the y-axis), in the code this function is called **motionprofileraw()**. The boundaries needed for each function within the piecewise (such as increasing acceleration from 0-0.5s, decreasing acceleration from 0.5-1s, keeping acceleration constant from 1-5.6s is calculated before the movement is executed using separate equations shown in **motionprofileconst()**).

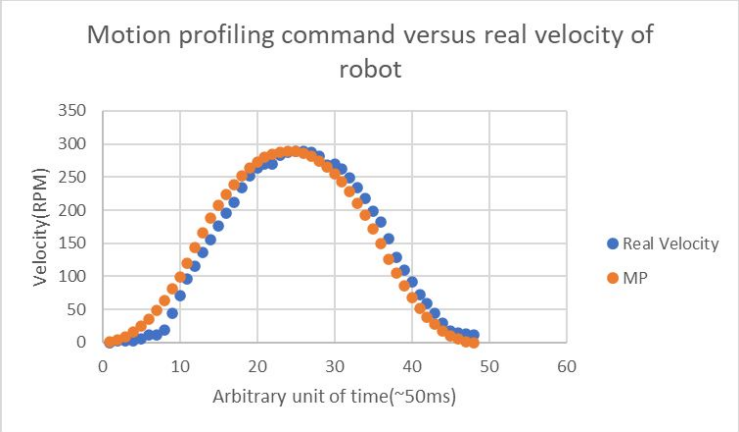
Motion Profiling Testing/Code

Test Observations

After testing, we found the robot sometimes jumps at the start when using PID, due to weight differences and large jerk. Therefore, it doesn't drive straight as well as when using motion profiling, where jerk is minimized.

By comparing the robot's actual velocity when following a PID profile versus a motion profile, we see that





We will use motion profiling in skills only, as in autonomous, speed as much as accuracy, and here speed is sacrificed for consistency's sake

Program

Here is the main function used for motion profiling. As shown, it is a while loop which keeps updating the robot's velocity until the previously calculated total time taken to finish the movement(Tt) is reached.

```
139  def autonomous():
140      #driving straight for 36 inches
141      s = 36
142      #converting max velocity from 600 RPM to inches/s since all calculations made in motion profile use inches
143      v = minutestoseconds(rotationtoinch(600))
144      #converting max acceleration from 400 RPM to inches/s^2
145      a = minutestoseconds(rotationtoinch(400))
146      #calculates all motion profile constants
147      motionprofileconst(s,v,a)
148      start = brain.timer.time(SECONDS) #takes note of start time
149      reftime = brain.timer.time(SECONDS) - start #calculates time to input into motion profiling velocity function
150      while reftime<=Tt:
151          #MOTION PROFILING
152          reftime = brain.timer.time(SECONDS)-start
153          vel = inchtorotation(secondstominutes(motionprofileraw(reftime, v, a)))
154          setright(vel)
155          setleft(vel)
156
157          reftime = brain.timer.time(SECONDS)-start
158      drivestop()
```

Motion Profiling Code

Program

Here is the function which calculates all the constants needed (total time to finish movement, total time needed to accelerate, etc.), for the piecewise function's boundaries:

```
201    def motionprofileconst(s, maxvel, maxaccel):
202        #sets up constants
203        global sgn, Yf, Ys, Yaux, Ya, Vw, To, Ta, tau, Tm, Tk, Ts, Tt, gamma
204        if s<0:
205            sgn = -1
206        else:
207            sgn = 1
208        Yf = abs(s)
209        Ys = Yf/2
210        Yaux = (1/2)*(1+gamma)*((maxvel*maxvel)/maxaccel)
211        if Ys<=Yaux:
212            Ya = Ys
213            Vw = math.sqrt((Ys*maxaccel)/((1/2)*(1+gamma)))
214        else:
215            Ya = Yaux
216            Vw = maxvel
217        To = Vw/maxaccel
218        Ta = To*(1+gamma)
219        tau = gamma*To
220        Tm = Ta/2
221        Tk = 2*((Ys-Ya)/maxvel)
222        Ts = Ta+(Tk/2)
223        Tt = 2*Ts
```

This function calculates the velocity at an exact timestamp:

```
225    def motionprofileraw(t, maxvel, maxaccel):
226        #outputs required velocity
227        global sgn, Yf, Ys, Yaux, Ya, Vw, To, Ta, tau, Tm, Tk, Ts, Tt, vel
228
229        if t <= 0:
230            vel = 0
231        if 0<t<tau: #until max accel reached
232            vel = (maxaccel/2)*((pow(t,2))/tau)
233        if tau<=t<=To: #until accel finished
234            vel = (maxaccel/2)*(t*2-tau)
235        if Tm<t<=Ts: #middle of accel ceiling to middle of movement(mostly from end of 1st deaccel)
236            vel = Vw - motionprofileraw(Ta-t, maxvel, maxaccel) #idk the excel sheet doesnt use this
237        if Ts<t:
238            vel = motionprofileraw(Tt-t, maxvel, maxaccel)#converting from inch/s to RPM
239        vel *= sgn
240        return vel
```

Odometry

Goal

We want to increase the speed of the autonomous while retaining accuracy, We implement a position tracking system in the robot to achieve this.

Requirements

- The robot should be able to track it's own position at all times
  - Including when it accidently umps up or accelerates quickly
- Need 10/10 accuracy

Explanation

High speed tends to cause overshoot or cause the robot to drive curved/not straight/jump when driving. Thus, by using position tracking, the robot has leeway with imprecise movement, and can simply recalculate it's trajectory if it goes off course during the autonomous or programming skills.

Inspiration was taken from the team that introduced odometry to VEX, E-Pilons 5225A: http://thepilons.ca/wp-content/uploads/2018/10/Tracking.pdf
And a very good video series: https://www.youtube.com/watch?v=GEZBYHVHmFQ

Code

The main function executes two different threads, one that uses odometry to calculate position(**thread_odom**), and one executes PIDs to get to a goal position(**thread_pathfinding**).

```
506  def autonomous():
507      inertial.calibrate()
508      wait(2, SECONDS)
509      resetposition(0,0,0) #reset x, y, and theta to 0
510      frDrive.set_stopping(BRAKE)
511      flDrive.set_stopping(BRAKE)
512      brDrive.set_stopping(BRAKE)
513      blDrive.set_stopping(BRAKE)
514
515      Thread(thread_odom)
516      Thread(thread_pathfinding)
```

Odometry

```
518  def thread_odom():
519      count = 0
520      global x_c, y_c, x_s, y_s, x, y
521      prevtime = brain.timer.time(MSEC)
522      while True:
523          count+=1
524          if count >=5:
525              print("x_s:", x_s, ", y_s:", y_s)
526              print("x_c:", x_c, ", y_c:", y_c)
527              print("x:", x, ", y:", y)
528              count = 0
529          measure_distance()
530          update_position_s()
531          update_position_c()
532          x = x_c*0.7 + x_s*0.3
533          y = y_c*0.7 + y_s*0.3
534          wait(10, MSEC)
```

The odometry thread first updates the robot's encoder values(**left_encoder_count** and **right_encoder_count**) using **measure_distance()**, then calculates the robot's position using those values paired with the angle of the robot, determined by the inertial sensor, to calculate final The odometry thread first updates the robot's encoder values, horizontal(**x_s**), and vertical(**y_s**) positions

```
276  def measure_distance():
277      global theta_s, left_encoder_count, right_encoder_count, back_encoder_count, l_encoder_count, prevflpos, prevfrpos
278      curflpos = avgleft()
279      curfrpos = avgright()
280      curlpos = encoder_l.position(TURNS)
281      curbpos = encoder_s.position(TURNS)
282      theta_s = (inertial.heading()*math.pi)/180
283      left_encoder_count = (curflpos-prevflpos)*36/84  #36/84 is ratio of gear from motor to wheels
284      right_encoder_count = (curfrpos-prevfrpos)*36/84
285      l_encoder_count = (curlpos-prevlpos)
286      back_encoder_count = (curbpos - prevbpos)
287      prevflpos = curflpos
288      prevfrpos = curfrpos
289      prevbpos = curbpos
290      prevlpos = curlpos
291      # for testing
292      total_count_left+= left_encoder_count
293      total_count_right+= right_encoder_count
```

```
346  def update_position_s():
347      global x_s,y_s,theta_s,prevtheta_s,left_encoder_count,right_encoder_count,wheel_circumference_rl,wheel_circumference_s
348      #// = to try
349
350      distance_left = get_wheel_distance(left_encoder_count, wheel_circumference_rl)
351      distance_right = get_wheel_distance(right_encoder_count, wheel_circumference_rl)
352      distance_back = get_wheel_distance(back_encoder_count, wheel_circumference_s)
353      distance_center = (distance_left+distance_right)/2
354      theta_change = theta_s-prevtheta_s
355
356      if(theta_change == 0):
357          dy = distance_center
358      else:
359          dy = 2*((distance_right/theta_change)+Tr)*(math.sin(theta_change/2))
360
361      beta = prevtheta_s + (theta_change/2)
362
363      x_s += dy*math.sin(beta)
364      y_s += dy*math.cos(beta)
365
366      prevtheta_s = theta_s
```

Odometry Testing

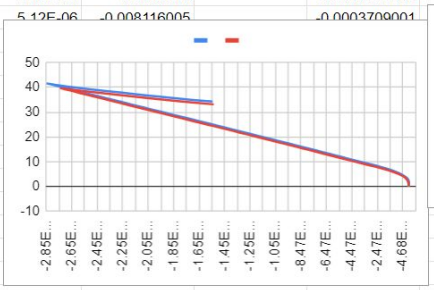
Testing Graphs

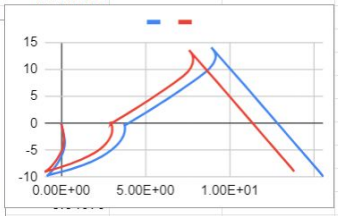
odom data

File Edit View Insert Format Data Tools Extensions Help

L22

	A	B	C	D	E	F	G	H	I	J	K
1											
2	3.92E-08	-0.0007103128		3.59E-08	-0.0006796474		5.72E-08	-0.0008575067			
3	1.04E-07	-0.001158028		1.03E-07	-0.001148828		-2.84E-05	0.04981467			
4	3.11E-07	-0.002001326		3.14E-07	-0.002010526		-8.70E-05	0.09759813			
5	7.98E-07	-0.00320341		8.35E-07	-0.003277006		-7.17E-05	0.09140182			
6	1.77E-06	-0.004770411		1.75E-06	-0.004739746		-0.0001046998	0.09920835			
7	3.33E-06	-0.006549003		3.42E-06	-0.0066318		-0.0002519658	0.1159709			
8											
9											
10											
11											
12											
13											
14											
15											
16											
17	6.54E-06	-0.009170893						-3.550614			
18	6.38E-06	-0.009060497						-4.159622			
19	6.45E-06	-0.009106494						-4.750193			
20	6.50E-06	-0.009143291		6.65E-06	-0.009250624		0.1093174	-5.342179			
21	6.40E-06	-0.009075828		6.57E-06	-0.009195425		0.03706706	-5.934093			
22	6.35E-06	-0.009039029		6.76E-06	-0.009327287		-0.06498771	-6.614783			





Observations

- Discrepancy between position calculated by tracking wheel and two side wheels
- Error for final position calculations is ~2-4 inches for horizontal position and ~0.5-1.5 inches for vertical position
- Overall wheel diameters likely need to be tuned further, and odometry pods improved

Final Code

```
#region VEXcode Generated Robot Configuration
from vex import *
import urandom

# Brain should be defined by default
brain=Brain()

# Robot configuration code
brDrive = Motor(Ports.PORT20, GearSetting.RATIO_6_1, False)
frDrive = Motor(Ports.PORT10, GearSetting.RATIO_6_1, False)
flDrive = Motor(Ports.PORT1, GearSetting.RATIO_6_1, True)
blDrive = Motor(Ports.PORT11, GearSetting.RATIO_6_1, True)
controller_1 = Controller(PRIMARY)
catapult = Motor(Ports.PORT5, GearSetting.RATIO_36_1, False)
wings = DigitalOut(brain.three_wire_port.h)
inertial = Inertial(Ports.PORT14)
encoder_s = Encoder(brain.three_wire_port.c)
mrDrive = Motor(Ports.PORT9, GearSetting.RATIO_6_1, True)
mlDrive = Motor(Ports.PORT2, GearSetting.RATIO_6_1, False)
intake = DigitalOut(brain.three_wire_port.a)
bumperE = Bumper(brain.three_wire_port.e)
hang = Motor(Ports.PORT16, GearSetting.RATIO_36_1, True)


# wait for rotation sensor to fully initialize
wait(30, MSEC)


def play_vexcode_sound(sound_name):
# Helper to make playing sounds from the V5 in VEXcode easier and
# keeps the code cleaner by making it clear what is happening.
print("VEXPlaySound:" + sound_name)
wait(5, MSEC)

# add a small delay to make sure we don't print in the middle of the REPL header
wait(200, MSEC)
# clear the console to make sure we don't have the REPL in the console
print("\033[2J")

#endregion VEXcode Generated Robot Configuration

# ------------------------------------------
#
#   Project:
#   Author:
#   Created:
#   Configuration:
#
# ------------------------------------------

# Library imports
from vex import *
# 47, 17, 3900
# Begin project code

```


Name: Carol Wang and Aleena Long

```
56  # CLASSES
57
58  class Button:
59  global row_to_pixel, column_to_pixel
60  textColour = Color.WHITE
61  def __init__(self, colour, borderWidth, borderColour, s, textX, textY, addWidth, add
  Height):
62  self.s = s
63  self.textX = textX
64  self.textY = textY
65  length = len(s)
66  #x and y of button are at center of word
67  #remember: row & column numbers start at 1, not at 0, that's why textX - 1 and textY
  z
68  x = ((textX-1) + length/2)*column_to_pixel
69  y = ((textY-1) + 0.5)*row_to_pixel
70  textWidth = length*column_to_pixel
71  textHeight = row_to_pixel
72  #all values that are going into a VEX function have to be rounded
73  self.width = addWidth + textWidth
74  self.height = addHeight + textHeight
75  self.x = round(x - self.width/2)
76  self.y = round(y - self.height/2)
77  self.colour = colour
78  self.borderWidth = borderWidth
79  self.borderColour = borderColour
80  def setColour(self, colour):
81  self.colour = colour
82  def setBorder(self, borderWidth, borderColour):
83  self.borderWidth = borderWidth
84  self.borderColour = borderColour
85  def setTextColour(self, colour):
86  self.textColour = colour
87  def setText(self, s):
88  self.s = s
89  def drawButton(self):
90  drawRect(self.colour, self.borderWidth, self.borderColour, self.x, self.y, self.widt
  h, self.height)
91  brain.screen.set_cursor(self.textY,self.textX)
92  brain.screen.set_pen_color(self.textColour)
93  brain.screen.print(self.s)
94  brain.screen.set_pen_color(Color.WHITE)
95  def drawRoundButton(self, coeff):
96  radius = round(coeff*self.height/2)
97  print(radius)
98  x = self.x + radius
99  width = self.width-2*radius
100 height = self.height-2*radius
101 drawRect(self.colour, self.borderWidth, self.borderColour, x, self.y, width, self.he
  ight)
102 y = self.y + radius #since radius is half of height, which is additional to y of rec
  t where circle y should start
103 brain.screen.set_fill_color(self.colour)
104 brain.screen.set_pen_width(self.borderWidth)
105 brain.screen.set_pen_color(self.borderColour)
106 brain.screen.draw_circle(x, y, radius)
```


Name: Carol Wang and Aleena Long

```
107  brain.screen.draw_circle(x, y+height, radius)
108  brain.screen.draw_circle(x+width, y, radius)
109  brain.screen.draw_circle(x+width, y+height, radius)
110  drawRect(self.colour, self.borderWidth, self.borderColour, x, y, -radius, height)
111  drawRect(self.colour, self.borderWidth, self.borderColour, x+width, y, radius, heigh
  t)
112  # drawRect(Color.GREEN, self.borderWidth, self.borderColour, x, y, width, -radius)
113  # drawRect(Color.GREEN, self.borderWidth, self.borderColour, x, y+height, width, rad
  ius)
114  brain.screen.set_cursor(self.textY,self.textX)
115  brain.screen.set_pen_color(self.textColour)
116  brain.screen.print(self.s)
117  brain.screen.set_pen_color(Color.WHITE)
118  brain.screen.set_pen_width(1)
119  def inBounds(self, x, y):
120  if(self.x<x<self.x+self.width and self.y<y<self.y+self.height):
121  return True
122  else:
123  return False
124
125
126  # ----------------------------------------
127  # main
128  # ----------------------------------------
129
130  brain.screen.set_font(FontType.MONO20)
131
132  #useful to convert percentage position to pixel position
133  #ex: 50*x_percent gets pixel position at 50% of screen width
134  x_percent = 480/100
135  y_percent = 240/100
136
137  #rows and columns change when brain font size changes
138  rows = 12
139  columns = 48
140  row_percent = rows/100
141  column_percent = columns/100
142  row_to_pixel = 240/rows
143  column_to_pixel = 480/columns
144
145  #to create dropdown menu
146  #autons, motors, sensor (all 6 letters, will all be centered)
147  menuOptions = ["AUTONS", "MOTORS", "SENSORS"]
148  menu = Button(Color(0x2596be), 0, Color.TRANSPARENT, menuOptions[0], round(10*column
  _percent), round(20*row_percent), round(10*x_percent), round(5*y_percent))
149
150  #button specs
151  #remember: row & column numbers start at 1, not at 0
152  incrementY = round(25*row_percent)
153
154  redWidth = round(10*x_percent)
155  redHeight = round(5*y_percent)
156  red = Button(Color.RED, 0, Color.TRANSPARENT, "RIGHT", round(10*column_percent), rou
  nd(40*row_percent), redWidth, redHeight)
157  redOptionsX = round(40*column_percent)
158  redOptionsY = round(25*row_percent)
```

```
159  redOptions = (Button(Color.RED, 0, Color.TRANSPARENT, "1", redOptionsX, redOptionsY,
   redWidth, redHeight), Button(Color.RED, 0, Color.TRANSPARENT, "2", redOptionsX, redOpti
   onsY+incrementY, redWidth, redHeight), Button(Color.RED, 0, Color.TRANSPARENT, "Slay",
   redOptionsX, redOptionsY+2*incrementY, redWidth, redHeight))
160
161  blueWidth = round(10*x_percent)
162  blueHeight = round(5*y_percent)
163  blue = Button(Color.BLUE, 0, Color.TRANSPARENT, "LEFT", round(10*column_percent), ro
   und(70*row_percent), blueWidth, blueHeight)
164  blueOptionsX = round(40*column_percent)
165  blueOptionsY = round(25*row_percent)
166  blueOptions = (Button(Color.BLUE, 0, Color.TRANSPARENT, "1", blueOptionsX, blueOptio
   nsY, blueWidth, blueHeight), Button(Color.BLUE, 0, Color.TRANSPARENT, "2", blueOptions
   X, blueOptionsY+incrementY, blueWidth, blueHeight), Button(Color.BLUE, 0, Color.TRANSPA
   RENT, "Slay", blueOptionsX, blueOptionsY+2*incrementY, blueWidth, blueHeight))
167
168  readyX = round(60*column_percent)
169  readyY = round(50*row_percent)
170  readyWidth = round(10*x_percent)
171  readyHeight = round(10*y_percent)
172  readyButton = Button(Color.YELLOW, 0, Color.TRANSPARENT, "READY", readyX, readyY, re
   adyWidth, readyHeight)
173
174  #updating
175  FPS = 24
176  f = 1000/FPS #milliseconds per update
177  prevTime = 0
178  curTime = 0
179
180  #vars
181  preAuton = True
182  screens = 0
183  ready = False
184  auton = 0 #1,2,3 for red, 4,5,6 for blue, 0 for default
185  autonColor = 0 #1 is red, 2 is blue
186
187  changeds = False
188  changing = False
189  screenChange = 0
190
191  calibrating = False
192  calibrateStart = 0
193  calibrated = False
194
195  unplugged = False
196
197  #AUTON VARIABLES
198  autonValue = 0
199  autonDesc = ["stay still", "autonRight", "autonRightWP", "autonLeft", "autonLeftWP"]
200  maxAuton = 4
201
202  def repaint():
203  global screens, changeds, changing, ready, screenChange, autonValue
204  #check vars
205  brain.screen.set_fill_color(Color.BLACK)
206  brain.screen.draw_rectangle(-1,-1, 482, 242)
```

```
207 if unplugged is not True:
208 controller_1.screen.set_cursor(3,1)
209 controller_1.screen.print("AUTON:", autonValue)
210 controller_1.screen.set_cursor(3,10)
211 if ready:
212 controller_1.screen.print("READY?:", "Y")
213 else:
214 controller_1.screen.print("READY?:", "N")
215
216 if(menu.inBounds(brain.screen.x_position(),brain.screen.y_position())):
217 if (changing is True and changeds is not True):
218 screens+=1
219 if screens is len(menuOptions):
220 #max screens is menuOptions length, if over max, restart screen count at 0
221 screens = 0
222 menu.setText(menuOptions[screens])
223 elif (changing is True and screenChange is not 0):
224 screens+=screenChange
225 if screens is len(menuOptions):
226 #max screens is menuOptions length, if over max, restart screen count at 0
227 screens = 0
228 elif screens is -1:
229 screens = len(menuOptions) - 1
230 menu.setText(menuOptions[screens])
231 screenChange = 0
232 if(screens is 0):
233 screenOne()
234 elif (screens is 1):
235 screenTwo()
236 elif (screens is 2):
237 screenThree()
238 if (brain.screen.pressing()):
239 changing = False
240 changeds = False
241 elif controller_1.buttonR1.pressing():
242 changing = False
243 changeds = False
244 screenChange = 1
245 elif controller_1.buttonL1.pressing():
246 changing = False
247 changeds = False
248 screenChange = -1
249 else:
250 changing = True
251 def screenOne():
252 global red, blue, menu, autonColor, auton, ready, changeds, calibrating, calibrateSt
  art, calibrated, autonValue
253 #track press
254 if(red.inBounds(brain.screen.x_position(),brain.screen.y_position())):
255 autonColor = 1
256 elif(blue.inBounds(brain.screen.x_position(),brain.screen.y_position())):
257 autonColor = 2
258 #render
259 if autonColor is 1:
260 if(redOptions[0].inBounds(brain.screen.x_position(),brain.screen.y_position())):
```


Name: Carol Wang and Aleena Long

```
auton = 1
ready = False
elif(redOptions[1].inBounds(brain.screen.x_position(),brain.screen.y_position())):
auton = 2
ready = False
elif(redOptions[2].inBounds(brain.screen.x_position(),brain.screen.y_position())):
auton = 3
ready = False
red.setBorder(3, Color.YELLOW)
red.drawButton()
blue.drawButton()
if auton < 4 and auton is not 0:
redOptions[auton-1].setColour(Color.WHITE)
redOptions[auton-1].setTextColour(Color.RED)
redOptions[auton-1].setBorder(3,Color.RED)
redOptions[0].drawButton()
redOptions[1].drawButton()
redOptions[2].drawButton()
red.setBorder(0, Color.TRANSPARENT)
if auton < 4 and auton is not 0:
redOptions[auton-1].setColour(Color.RED)
redOptions[auton-1].setTextColour(Color.WHITE)
redOptions[auton-1].setBorder(0,Color.TRANSPARENT)
if auton != 0:
if readyButton.inBounds(brain.screen.x_position(),brain.screen.y_position()):
if (changing is True and changeds is not True):
readyButton.setText("LOADING...")
inertial.calibrate()
hang.set_position(0, TURNS)
calibrateStart = brain.timer.time(MSEC)
calibrating = True
changeds = True
if calibrating:
if (brain.timer.time(MSEC) - calibrateStart) > 2000:
readyButton.setText("READY")
controller_1.rumble("...")
autonValue = auton
ready = True
calibrating = False

if ready:
readyButton.setColour(Color.WHITE)
readyButton.setTextColour(Color.BLACK)
else:
readyButton.setColour(Color.YELLOW)
readyButton.setTextColour(Color.WHITE)
readyButton.drawButton()

elif autonColor is 2:
if(blueOptions[0].inBounds(brain.screen.x_position(),brain.screen.y_position())):
auton = 4
ready = False
elif(blueOptions[1].inBounds(brain.screen.x_position(),brain.screen.y_position())):
auton = 5
ready = False
```


Name: Carol Wang and Aleena Long

```
316 elif(blueOptions[2].inBounds(brain.screen.x_position(),brain.screen.y_position())):
317 auton = 6
318 ready = False
319
320 blue.setBorder(3, Color.YELLOW)
321 red.drawButton()
322 blue.drawButton()
323 if auton > 3:
324 blueOptions[auton-4].setColour(Color.WHITE)
325 blueOptions[auton-4].setTextColour(Color.BLUE)
326 blueOptions[auton-4].setBorder(3,Color.BLUE)
327 blueOptions[0].drawButton()
328 blueOptions[1].drawButton()
329 blueOptions[2].drawButton()
330 blue.setBorder(0, Color.TRANSPARENT)
331 if auton > 3:
332 blueOptions[auton-4].setColour(Color.BLUE)
333 blueOptions[auton-4].setTextColour(Color.WHITE)
334 blueOptions[auton-4].setBorder(0,Color.TRANSPARENT)
335 if auton != 0:
336 if readyButton.inBounds(brain.screen.x_position(),brain.screen.y_position()):
337 readyButton.setText("LOADING...")
338 inertial.calibrate()
339 calibrating = True
340 if calibrating:
341 wait(2, SECONDS)
342 controller_1.rumble("...")
343 ready = True
344 autonValue = auton
345 calibrating = False
346 if ready:
347 readyButton.setColour(Color.WHITE)
348 readyButton.setTextColour(Color.BLACK)
349 else:
350 readyButton.setColour(Color.YELLOW)
351 readyButton.setTextColour(Color.WHITE)
352 readyButton.drawButton()
353 else:
354 red.drawButton()
355 blue.drawButton()
356 menu.drawButton()
357 brain.screen.render()
358 resetPen()
359
360 def screenTwo():
361 global menu
362 #render
363 menu.drawButton()
364
365 brain.screen.set_cursor(row_percent*30,column_percent*5)
366 brain.screen.print("frDrive(10):", frDrive.temperature(PERCENT), getInstalled(10))
367 brain.screen.set_cursor(row_percent*40,column_percent*5)
368 brain.screen.print("flDrive(1):", flDrive.temperature(PERCENT), getInstalled(1))
369 brain.screen.set_cursor(row_percent*50,column_percent*5)
370 brain.screen.print("brDrive(20):", brDrive.temperature(PERCENT), getInstalled(20))
```

```
    brain.screen.set_cursor(row_percent*60,column_percent*5, )
372 brain.screen.print("blDrive(11):", blDrive.temperature(PERCENT), getInstalled(11))
373 brain.screen.set_cursor(row_percent*70,column_percent*5)
374 brain.screen.print("mrDrive(2):", mrDrive.temperature(PERCENT), getInstalled(2))
375 brain.screen.set_cursor(row_percent*80,column_percent*5)
376 brain.screen.print("mlDrive(9):", mlDrive.temperature(PERCENT), getInstalled(9))
377
378 brain.screen.set_cursor(row_percent*40,column_percent*60)
379 brain.screen.print("cata(5):", catapult.temperature(PERCENT), getInstalled(5))
380 brain.screen.set_cursor(row_percent*60,column_percent*60)
381 brain.screen.print("hang(16):", hang.position(TURNS), getInstalled(16))
382
383 brain.screen.render()
384 resetPen()
385 def screenThree():
386 global ready
387 #render
388 menu.drawButton()
389 brain.screen.set_cursor(row_percent*50,column_percent*10)
390 brain.screen.print("INERTIAL:", inertial.heading())
391
392 brain.screen.set_cursor(row_percent*50,column_percent*50)
393 if bumperE.pressing():
394 brain.screen.print("BUMPER:", "Y")
395 else:
396 brain.screen.print("BUMPER:", "N")
397 brain.screen.set_cursor(row_percent*70,column_percent*10)
398 brain.screen.print("AUTON:", autonValue)
399
400 brain.screen.set_cursor(row_percent*70,column_percent*50)
401 brain.screen.print("READY?:", ready)
402
403 brain.screen.render()
404 resetPen()
405 def pre_autonomous():
406 global prevTime, curTime, FPS, preAuton
407 while preAuton is True:
408 curTime = brain.timer.time(MSEC)
409 if((curTime-prevTime)>=f):
410 prevTime = curTime
411 repaint()
412 checkInstalled()
413 if controller_1.buttonA.pressing():
414 preAuton = False
415 def drawRect(colour, borderWidth, borderColour, x, y, width, height):
416 brain.screen.set_fill_color(colour)
417
418 brain.screen.set_pen_width(borderWidth)
419 brain.screen.set_pen_color(borderColour)
420 brain.screen.draw_rectangle(x, y, width, height)
421
422 def checkInstalled():
423 global unplugged
424 unplugged = False
425 count = 0
```

```
426 s = ""
427 if flDrive.installed() is False:
428 s = "flDrive(1)"
429 count+=1
430 if frDrive.installed() is False:
431 s = "frDrive(10)"
432 count+=1
433 if blDrive.installed() is False:
434 s = "blDrive(11)"
435 count+=1
436 if brDrive.installed() is False:
437 s = "brDrive(20)"
438 count+=1
439 if mlDrive.installed() is False:
440 s = "mlDrive(2)"
441 count+=1
442 if mrDrive.installed() is False:
443 s = "mrDrive(9)"
444 count+=1
445 if catapult.installed() is False:
446 s = "cata(5)"
447 count+=1
448 # if hang.installed() is False:
449 # s = "hang(16)"
450 # count+=1
451 if inertial.installed() is False:
452 s = "inertial(14)"
453 count+=1
454 if count > 0:
455 controller_1.screen.set_cursor(3, 0)
456 controller_1.screen.print(s+" unplugged")
457 if preAuton:
458 controller_1.rumble(".-")
459 wait(3000, MSEC)
460 controller_1.screen.clear_row(3)
461 unplugged = True
462 def getInstalled(portNumber):
463 if(portNumber is 1):
464 if flDrive.installed():
465 return "Y"
466 else:
467 return "N"
468 if(portNumber is 10):
469 if frDrive.installed():
470 return "Y"
471 else:
472 return "N"
473 if(portNumber is 11):
474 if blDrive.installed():
475 return "Y"
476 else:
477 return "N"
478 if(portNumber is 20):
479 if brDrive.installed():
480 return "Y"
```

```
    else:
482 return "N"
483 if(portNumber is 2):
484 if mlDrive.installed():
485 return "Y"
486 else:
487 return "N"
488 if(portNumber is 9):
489 if mrDrive.installed():
490 return "Y"
491 else:
492 return "N"
493 if(portNumber is 14):
494 if inertial.installed():
495 return "Y"
496 else:
497 return "N"
498 if(portNumber is 16):
499 if hang.installed():
500 return "Y"
501 else:
502 return "N"
503 if(portNumber is 5):
504 if catapult.installed():
505 return "Y"
506 else:
507 return "N"
508 def drawRoundButton(colour, borderWidth, borderColour, s, textX, textY, rect):
509 drawRect(colour, borderWidth, borderColour, rect[0], rect[1], rect[2], rect[3])
510 radius = round(rect[3]/2) #idk for some reason radius seems to be 1 px too big
511 y = rect[1] + radius #since radius is half of height, which is additional to y of re
  ct where circle y should start
512 brain.screen.set_fill_color(colour)
513 brain.screen.set_pen_width(borderWidth)
514 brain.screen.set_pen_color(borderColour)
515 brain.screen.draw_circle(rect[0], y, radius)
516 brain.screen.draw_circle(rect[0]+rect[2], y, radius)
517 drawRect(colour, borderWidth, colour, rect[0], rect[1]+borderWidth, rect[2], rect[3
  ]-(borderWidth*2))
518 brain.screen.set_cursor(textY,textX)
519 brain.screen.print(s)
520
521 def resetPen():
522 brain.screen.set_fill_color(Color.TRANSPARENT)
523 brain.screen.set_pen_width(1)
524 brain.screen.set_pen_color(Color.WHITE)
525
526
527
528 # def pre_autonomous():
529 # global preAuton
530 # inertial.calibrate()
531 # wait(2, SECONDS)
532 # controller_1.screen.set_cursor(3,1)
533 # controller_1.screen.print("CALIBRATED")
534 # wait(2, SECONDS)
```

```
# while preAuton:
# chooseAuton()
# if ready:
# return
def chooseAuton():
global screens, changeds, changing, autonValue, maxAuton, ready, preAuton
#check vars
if controller_1.buttonR2.pressing():
if changeds is False:
autonValue += 1
if autonValue > maxAuton:
autonValue = 0
changeds = True
ready = False
else:
changeds = False
if controller_1.buttonL2.pressing():
controller_1.screen.clear_row(3)
controller_1.screen.set_cursor(3,1)
controller_1.screen.print(autonDesc[autonValue])

brain.screen.set_cursor(row_percent*50,column_percent*10)
brain.screen.print("INERTIAL:", inertial.heading())

brain.screen.set_cursor(row_percent*50,column_percent*40)
brain.screen.print("AUTON:", autonValue)
brain.screen.set_cursor(row_percent*60,column_percent*40)
brain.screen.print(autonDesc[autonValue])

brain.screen.set_cursor(row_percent*50,column_percent*60)
brain.screen.print("READY?:", ready)

else:
brain.screen.set_cursor(row_percent*50,column_percent*10)
brain.screen.print("INERTIAL:", inertial.heading())

brain.screen.set_cursor(row_percent*50,column_percent*40)
brain.screen.print("AUTON:", autonValue)

brain.screen.set_cursor(row_percent*50,column_percent*60)
brain.screen.print("READY?:", ready)

controller_1.screen.set_cursor(3,1)
controller_1.screen.print("AUTON:", autonValue)
controller_1.screen.set_cursor(3,10)
controller_1.screen.print("READY?:", ready)
brain.screen.set_fill_color(Color.BLACK)
brain.screen.draw_rectangle(-1,-1, 482, 242)
brain.screen.render()
resetPen()

def autonomous():
global preAuton, autonValue
preAuton = False
brain.screen.clear_screen()
```

```
590 brain.screen.print("autonomous")
591 controller_1.screen.clear_row(3)
592 if ready is False:
593 autonValue = 0
594 if autonValue is 0:
595 wait(1, SECONDS)
596 elif autonValue is 1:
597 autonRight()
598 elif autonValue is 2:
599 autonRightWP()
600 elif autonValue is 3:
601 autonLeft()
602 elif autonValue is 4:
603 autonLeftWP()
604
605 def autonRight():
606 wait(250, MSEC)
607 def autonRightWP():
608 #drive forward
609 wait(250, MSEC)
610 def autonLeft():
611 wait(200, MSEC)
612 def autonLeftWP():
613 wait(200, MSEC)
614
615 def user_control():
616 global preAuton
617
618 preAuton = False
619 brain.screen.set_fill_color(Color.BLACK)
620 brain.screen.draw_rectangle(-1,-1, 482, 242)
621 brain.screen.print("user-control")
622 brain.screen.render()
623 controller_1.screen.clear_row(3)
624
625 brain.screen.set_fill_color(Color.BLACK)
626 brain.screen.draw_rectangle(-1,-1, 482, 242)
627 brain.screen.print("user-control")
628 brain.screen.render()
629
630
631 #all initial velocity/stopping/variable declarations
632 bumperTime = 0
633 bumperDown = False
634 printed = False #for testing purposes
635 hang.set_stopping(HOLD)
636 catapult.set_velocity(100, PERCENT)
637 hang.set_velocity(50, PERCENT)
638
639 while True:
640 checkInstalled()
641
642 fbSpeed = controller_1.axis3.position()
643 lrSpeed = controller_1.axis1.position()
644
```

```
#DRIVECODE
if fbSpeed<=5 and fbSpeed>=-5:
fbSpeed = 0
if lrSpeed<=5 and lrSpeed>=-5:
lrSpeed = 0
if lrSpeed != 0 or fbSpeed != 0:
# if lrSpeed > fbSpeed:
# lrSpeed += skim(fbSpeed)
frDrive.set_velocity((fbSpeed - lrSpeed), PERCENT)
flDrive.set_velocity((fbSpeed + lrSpeed), PERCENT)
brDrive.set_velocity((fbSpeed - lrSpeed), PERCENT)
blDrive.set_velocity((fbSpeed + lrSpeed), PERCENT)
mrDrive.set_velocity((fbSpeed - lrSpeed), PERCENT)
mlDrive.set_velocity((fbSpeed + lrSpeed), PERCENT)
frDrive.spin(FORWARD)
flDrive.spin(FORWARD)
blDrive.spin(FORWARD)
brDrive.spin(FORWARD)
mrDrive.spin(FORWARD)
mlDrive.spin(FORWARD)
elif lrSpeed != 0 and fbSpeed == 0:
# lrSpeed = lrSpeed*0.5
frDrive.set_velocity(-lrSpeed, PERCENT)
flDrive.set_velocity(lrSpeed, PERCENT)
brDrive.set_velocity(-lrSpeed, PERCENT)
blDrive.set_velocity(lrSpeed, PERCENT)
mrDrive.set_velocity(-lrSpeed, PERCENT)
mlDrive.set_velocity(lrSpeed, PERCENT)
frDrive.spin(FORWARD)
flDrive.spin(FORWARD)
blDrive.spin(FORWARD)
brDrive.spin(FORWARD)
mlDrive.spin(FORWARD)
mrDrive.spin(FORWARD)
else:
drivestop1()

#OTHER MOTIONS
if controller_1.buttonA.pressing():
#kicker start
catapult.spin(FORWARD)
drivesetstop(HOLD)
elif controller_1.buttonB.pressing():
#kicker stop
catapult.set_stopping(COAST)
catapult.stop()
drivesetstop(COAST)
if controller_1.buttonR2.pressing():
#expand wings
wings.set(True)
elif controller_1.buttonL2.pressing():
#contract wings
wings.set(False)
if controller_1.buttonX.pressing():
#sends signal for catapult to reverse until it hits bumper sensor
```

```
  0 bumperDown = True
701 if bumperDown:
702 #when bumper is pressed, stop catapult
703 if (bumperE.pressing() is 0):
704 catapult.spin(FORWARD)
705 else:
706 bumperDown = False
707 catapult.set_stopping(HOLD)
708 catapult.stop()
709 bumperTime = brain.timer.time(MSEC)
710 if (brain.timer.time(MSEC) - bumperTime)>10000:
711 catapult.set_stopping(COAST)
712 if controller_1.buttonUp.pressing():
713 hang.spin(FORWARD)
714 bumperDown = True
715 elif controller_1.buttonDown.pressing():
716 catapult.set_stopping(COAST)
717 hang.spin(REVERSE)
718 else:
719 hang.stop()
720
721 #FOR TESTING PURPOSES(for auton)
722 if controller_1.buttonY.pressing():
723 if printed is False:
724 print("degrees:", inertial.heading(DEGREES))
725 print("motor turns:", avgturns())
726 # print("lr turns:", encoder_s.position(TURNS))
727 # print("br motor turns:", brDrive.position(TURNS))
728 # print("bl motor turns:", blDrive.position(TURNS))
729 printed = True
730 if controller_1.buttonR1.pressing():
731 intake.set(True)
732 elif controller_1.buttonL1.pressing():
733 intake.set(False)
734 printed = False
735
736 #USER-CONTROL FUNCTIONS
737 def skim(v):
738 # gain determines how much to skim off the top
739 if (v > 0):
740 return -(v * 0.1)
741 elif (v < 0):
742 return (v * 0.1)
743 return 0
744 def drivestop():
745 frDrive.stop()
746 flDrive.stop()
747 brDrive.stop()
748 blDrive.stop()
749 mrDrive.stop()
750 mlDrive.stop()
751 def drivestop1():
752 drivesetstop(COAST)
753 frDrive.stop()
754 flDrive.stop()
```

```
brDrive.stop()
blDrive.stop()
mrDrive.stop()
mlDrive.stop()
wait(35, MSEC)
drivesetstop(BRAKE)
def drivestop2():
drivesetstop(COAST)
frDrive.stop()
flDrive.stop()
brDrive.stop()
blDrive.stop()
mrDrive.stop()
mlDrive.stop()
wait(45, MSEC)
drivesetstop(BRAKE)

#AUTONOMOUS FUNCTIONS
#velocities
def drivesetstop(userinput):
frDrive.set_stopping(userinput)
flDrive.set_stopping(userinput)
brDrive.set_stopping(userinput)
blDrive.set_stopping(userinput)
mrDrive.set_stopping(userinput)
mlDrive.set_stopping(userinput)

def drivesetpercent(userinput):
frDrive.set_velocity(userinput,PERCENT)
flDrive.set_velocity(userinput,PERCENT)
brDrive.set_velocity(userinput,PERCENT)
blDrive.set_velocity(userinput,PERCENT)
mrDrive.set_velocity(userinput,PERCENT)
mlDrive.set_velocity(userinput,PERCENT)
#motions
def driveforward():
frDrive.spin(FORWARD)
flDrive.spin(FORWARD)
blDrive.spin(FORWARD)
brDrive.spin(FORWARD)
mlDrive.spin(FORWARD)
mrDrive.spin(FORWARD)

def driveforwardpercent(userinput):
frDrive.set_velocity(userinput,PERCENT)
flDrive.set_velocity(userinput,PERCENT)
brDrive.set_velocity(userinput,PERCENT)
blDrive.set_velocity(userinput,PERCENT)
mrDrive.set_velocity(userinput,PERCENT)
mlDrive.set_velocity(userinput,PERCENT)
frDrive.spin(FORWARD)
flDrive.spin(FORWARD)
blDrive.spin(FORWARD)
brDrive.spin(FORWARD)
mlDrive.spin(FORWARD)
```

```
810 mrDrive.spin(FORWARD)
811
812 #turning
813 def turnrightpercent(userinput):
814 #negative right and positive left motor velocities, to reverse right and go fwd with
  left
815 frDrive.set_velocity(-userinput,PERCENT)
816 flDrive.set_velocity(userinput,PERCENT)
817 mrDrive.set_velocity(-userinput,PERCENT)
818 mlDrive.set_velocity(userinput,PERCENT)
819 brDrive.set_velocity(-userinput,PERCENT)
820 blDrive.set_velocity(userinput,PERCENT)
821 frDrive.spin(FORWARD)
822 mlDrive.spin(FORWARD)
823 flDrive.spin(FORWARD)
824 brDrive.spin(FORWARD)
825 mrDrive.spin(FORWARD)
826 blDrive.spin(FORWARD)
827
828 def turnleftpercent(userinput):
829 #negative left and positive right motor velocities, to reverse right and go fwd with
  left
830 frDrive.set_velocity(userinput,PERCENT)
831 flDrive.set_velocity(-userinput,PERCENT)
832 mrDrive.set_velocity(userinput,PERCENT)
833 mlDrive.set_velocity(-userinput,PERCENT)
834 brDrive.set_velocity(userinput,PERCENT)
835 blDrive.set_velocity(-userinput,PERCENT)
836 frDrive.spin(FORWARD)
837 flDrive.spin(FORWARD)
838 brDrive.spin(FORWARD)
839 blDrive.spin(FORWARD)
840 mrDrive.spin(FORWARD)
841 mlDrive.spin(FORWARD)
842
843 #PIDs
844 def pidturn1(setpoint, curposition, start):
845 #PID for turning(to a point)
846 #if it turns over, will correct itself
847 #when setpoint > curposition, will turn right. otherwise, turn left.
848 integer = abs(setpoint - start)
849 Kp = 1/integer*9
850 Ki = 0
851 Kd = 0
852 global integralturn
853 global lasterrorturn
854 error = setpoint - curposition
855 # integralturn += error
856 # derivative = error-lasterrorturn
857 # lasterrorturn = error
858 #2.8 is the lowest possible voltage to still move in turn
859 turnpower = error*Kp#+integralturn*Ki+derivative*Kd
860 if 0 <=turnpower < 3:
861 turnpower = 3
862 if -3 <turnpower<0:
863 turnpower = -3
```

```
864 return turnpower
865 def pidturn2(setpoint, curposition):
866 global lasterrorturn
867 #velocity PID turning
868 #set right motors to reverse, left to forward
869 #if setpoint>curposition, turns right. else, turns left
870 Kp = 0.46
871 error = setpoint - curposition
872 turnpower = error*Kp
873 if turnpower> 100:
874 #makes sure turnpower not over 100% rpm
875 turnpower = 100
876 if 0<turnpower<2:
877 turnpower=2
878 elif 0>turnpower>-2:
879 turnpower=-2
880 if -1<error<1:
881 turnpower = 0
882 return turnpower
883 def pidturn3(setpoint, curposition):
884 global lasterrorturn
885 #velocity PID turning
886 #set right motors to reverse, left to forward
887 #if setpoint>curposition, turns right. else, turns left
888 Kp = 0.7
889 error = setpoint - curposition
890 turnpower = error*Kp
891 if turnpower> 100:
892 #makes sure turnpower not over 100% rpm
893 turnpower = 100
894 if 0<turnpower<5:
895 turnpower=5
896 elif 0>turnpower>-5:
897 turnpower=-5
898 if -1<error<1:
899 turnpower = 0
900 return turnpower
901
902 def pidforward(setpoint, curposition):
903 #velocity
904 #go to motor.turns value inputted as "setpoint"
905 #if setpoint > curposition, drives forward. If not, reverses.
906 Kp = 39 #tuning notes: 38 for smooth stop, 40 for fast stop
907 error = setpoint - curposition
908 drivepower = error*Kp
909 if drivepower> 100 or drivepower < -100:
910 #makes sure drivepower not over 100% rpm
911 if drivepower > 0:
912 coeff = 1
913 else:
914 coeff = -1
915 drivepower = 100*coeff
916 if 0 <=drivepower < 9:
917 drivepower = 9
918 if -9 <drivepower <0:
```

```
drivepower = -9
if -0.05<error<0.05:
drivepower=0
return drivepower
def pidforwardslow(setpoint, curposition):
#same as piddrive() but slower
Kp = 40
error = setpoint - curposition
drivepower = error*Kp
if drivepower> 60 or drivepower < -60:
#makes sure drivepower not over 100% rpm
if drivepower > 0:
coeff = 1
else:
coeff = -1
drivepower = 60*coeff
if 0 <=drivepower < 9:
drivepower = 9
if -9 <drivepower <0:
drivepower = -9
if -0.05<error<0.05:
drivepower=0
return drivepower
def piddriveget(setpoint, curposition):
#piddrive() but intake turns on while driving
Kp = 39
error = setpoint - curposition
drivepower = error*Kp
if drivepower> 100 or drivepower < -100:
if drivepower > 0:
coeff = 1
else:
coeff = -1
drivepower = 100*coeff
if 0 <=drivepower < 9:
drivepower = 9
if -9 <drivepower <0:
drivepower = -9
if -0.05<error<0.05:
drivepower=0
if -0.1<error<0.1:
intake.set(True)
return drivepower
def piddriveunget(setpoint, curposition):
#piddrive() but intake turns off while driving
Kp = 39
error = setpoint - curposition
drivepower = error*Kp
if drivepower> 100 or drivepower < -100:
if drivepower > 0:
coeff = 1
else:
coeff = -1
drivepower = 100*coeff
if 0 <=drivepower < 9:
```

```
974  drivepower = 9
975  if -9 <drivepower <0:
976  drivepower = -9
977  if -0.05<error<0.05:
978  drivepower=0
979  if -1<error<1:
980  intake.set(False)
981  return drivepower
982  def piddrivewingopen(setpoint, curposition, wingdistance):
983  #piddrive() but wings open while driving
984  Kp = 39
985  error = setpoint - curposition
986  drivepower = error*Kp
987  if drivepower> 100 or drivepower < -100:
988  if drivepower > 0:
989  coeff = 1
990  else:
991  coeff = -1
992  drivepower = 100*coeff
993  if 0 <=drivepower < 9:
994  drivepower = 9
995  if -9 <drivepower <0:
996  drivepower = -9
997  if -0.05<error<0.05:
998  drivepower=0
999  if abs(error)<(setpoint-wingdistance):
1000 wings.set(True)
1001 return drivepower
1002 def piddrivewingclose(setpoint, curposition, wingdistance):
1003 #piddrive() but wings close while driving
1004 Kp = 39
1005 error = setpoint - curposition
1006 drivepower = error*Kp
1007 if drivepower> 100 or drivepower < -100:
1008 if drivepower > 0:
1009 coeff = 1
1010 else:
1011 coeff = -1
1012 drivepower = 100*coeff
1013 if 0 <=drivepower < 9:
1014 drivepower = 9
1015 if -9 <drivepower <0:
1016 drivepower = -9
1017 if -0.05<error<0.05:
1018 drivepower=0
1019 if abs(error)<(setpoint-wingdistance):
1020 wings.set(False)
1021 return drivepower
1022
1023 #SKILLS FUNCTIONS
1024 def avgturns():
1025 avg = (frDrive.position(TURNS) + flDrive.position(TURNS) + brDrive.position(TURNS) +
  blDrive.position(TURNS))/4
1026 return avg
1027 def anglewrapdeg(deg):
```

```
1028 #converts angles above 180 to a negative angle under 180 and vice versa
1029 if(deg>180):
1030 deg = deg - 360
1031 # elif(deg<-180):
1032 # deg = deg + 360
1033 return deg
1034
1035 #Aleena
1036 def fullStop():
1037 frDrive.stop()
1038 mrDrive.stop()
1039 brDrive.stop()
1040 flDrive.stop()
1041 mlDrive.stop()
1042 blDrive.stop()
1043
1044 def forward():
1045 frDrive.spin(FORWARD)
1046 mrDrive.spin(FORWARD)
1047 brDrive.spin(FORWARD)
1048 flDrive.spin(FORWARD)
1049 mlDrive.spin(FORWARD)
1050 blDrive.spin(FORWARD)
1051
1052 def reverse():
1053 frDrive.spin(REVERSE)
1054 mrDrive.spin(REVERSE)
1055 brDrive.spin(REVERSE)
1056 flDrive.spin(REVERSE)
1057 mlDrive.spin(REVERSE)
1058 blDrive.spin(REVERSE)
1059
1060 def turnLeft():
1061 frDrive.spin(FORWARD)
1062 mrDrive.spin(FORWARD)
1063 brDrive.spin(FORWARD)
1064 flDrive.spin(REVERSE)
1065 mlDrive.spin(REVERSE)
1066 blDrive.spin(REVERSE)
1067
1068 def turnRight():
1069 frDrive.spin(REVERSE)
1070 mrDrive.spin(REVERSE)
1071 brDrive.spin(REVERSE)
1072 flDrive.spin(FORWARD)
1073 blDrive.spin(FORWARD)
1074 mlDrive.spin(FORWARD)
1075
1076 def anglewrap(deg):
1077 if deg > 180:
1078 deg = deg - 360
1079 else:
1080 deg = deg + 360
1081 return deg
1082
```

```
1083 comp = Competition(user_control, autonomous)
1084 pre_autonomous()
```


Name: Carol Wang and Aleena Long

Controller/Port Configurations

FLDRIVE 11

MLDRIVE 12

N/A 13

INERTIAL (sensor) 14

BLDRIVE 15

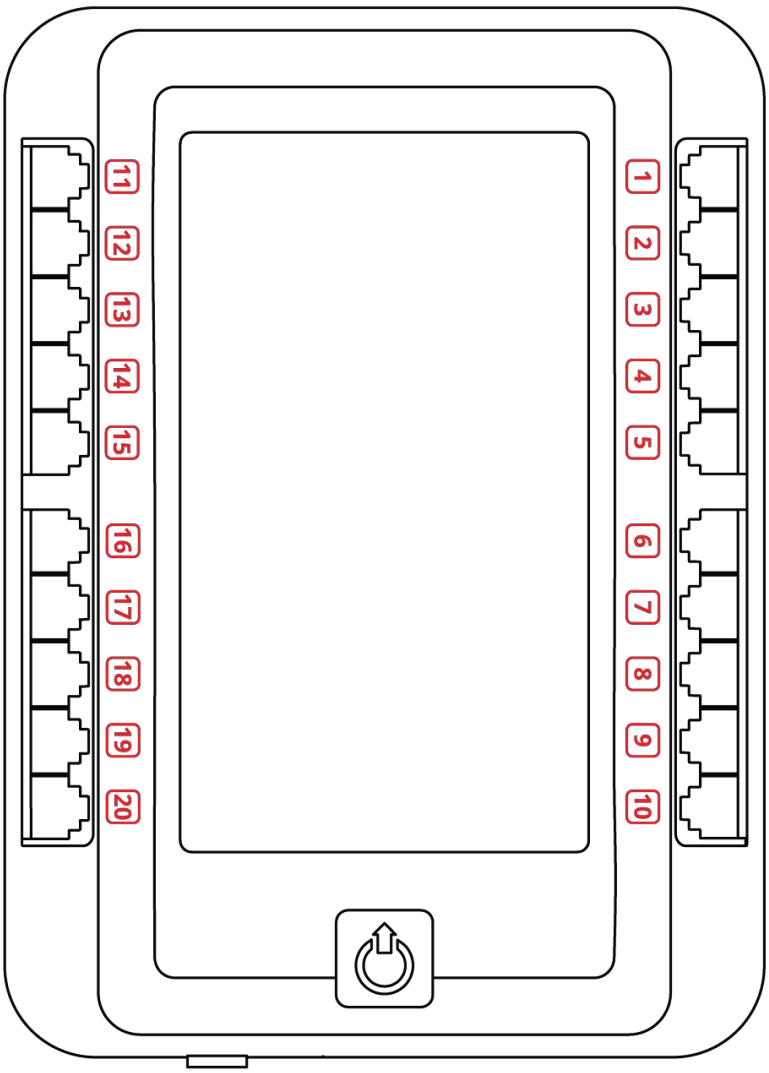
BRDRIVE 16

N/A 17

N/A 18

MRDRIVE 19

FRDRIVE 20



INTAKE 1

N/A 2

N/A 3

N/A 4

N/A 5

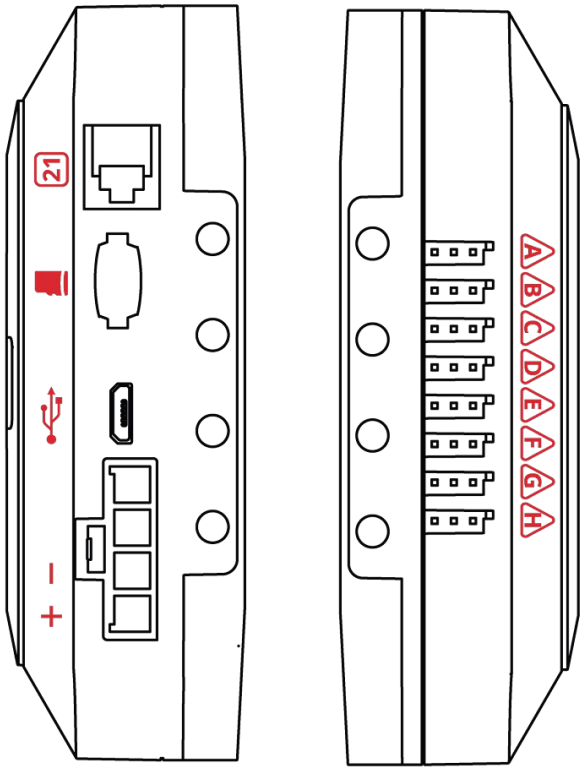
N/A 6

N/A 7

ROTATION (sensor) 8

N/A 9

CATAPULT 10



WINGS (pneumatics) A

N/A B

N/A C

N/A D

BUMPER E

N/A F

N/A G

N/A H

🔷KEY:

- Ports connect to motors, unless otherwise specified between brackets “()”
- For motors in the drivetrain, they follow the format “ABDRIVE”

“A” is:

- F = front, B = back, M = middle

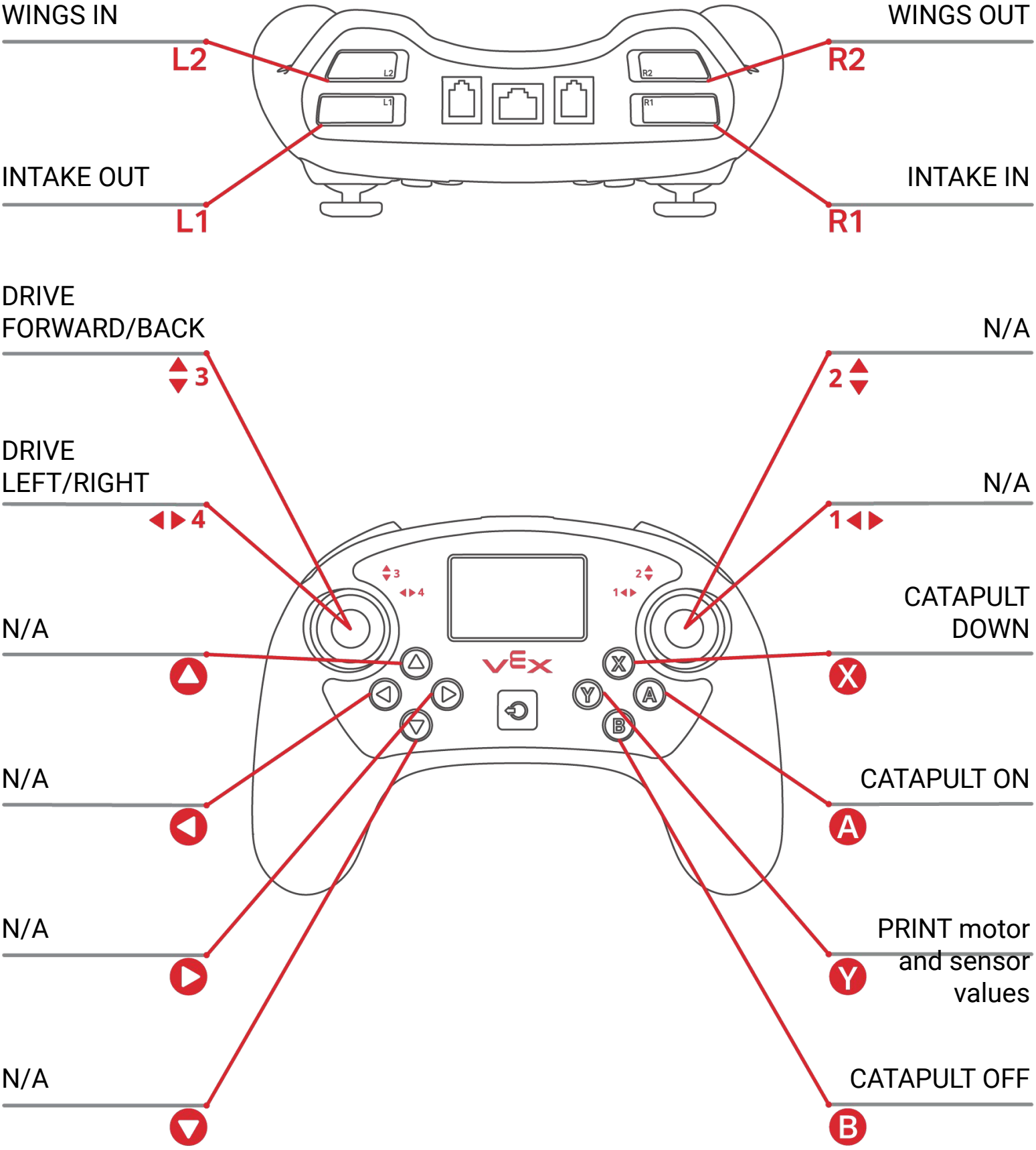
“B” is:

- L = left, R = right

For example: FRDRIVE indicates the front-right chassis motor

Project Port configurations

Name Carol Wang Date 02/16/2024 Page 68



Project Controller Configurations

Name Carol Wang Date 02/16/2024 Page 69