Pre-Scrimmage

RGB that stands out
Something special about our robot is RGB, which is something Alex has wanted to do since he started VEX, even though our coach says "it's a waste of time" at least we lose in style. There are two primary types of rgb strips, a 4 pad and a 3 pad. How the 4 pad works is there is 1 ground pad an 1 pad for each color (r,g,b). this means the whole strip will be that color, how rgb works is it mixes the primary colors, red, green, and blue ~~strip~~ spot ~~we~~ that input varying voltages but a vex brain can not do that. The old 3 wire motors also needed a analog signal too but used the 29 Motor Controller This converts a Pulse Width Modulation signal (PWM) into a analog. PWM is varying lengths of very fast on and off in a signal which the adi port can send. The issue is we need 1 for each color which takes up a lot of room and port is also ~~is no longer~~ legal because of <R6.
The other option is a adressable WS2811 led strip where using a pwm we can control every individual led but it uses a special pwm protocol. Since ~~it's~~ essetialy turning a ~~servo~~ port on and off we can implement this but the kernel makes it so that all user programs are run on CPU 1 thread.
(continued on pg 5a)

Scrimmage
We switched to tall bot to give us more room and also de scoring was not effective because of no double zone.
We relized 2 essential things we needed to improve was ~~puncher~~ wings and intake, new more reliable wings and faster intake. (show a diagram showing how four wire vex redirected wings)

Formosa 4/12

We travelled to taipei to compete in formosa, met many teams, and learnt lots from matches. We did okay during most of our matches.

On issue we encountered was that the sleds were unreliable and not smooth, this was because it had a very steep angle. We fixed this with our next iteration.

Even with lodite, our collar locks still sometimes fell out and it is very anoying to ~~re~~ retighten ~~it~~. The reason is that the two collar locks are in the corners. We overcame this by using spacers to move it to the center and easily ~~accessible~~ accessible

Another issue was bracing, the 4t metal could bend in and out in the front which caused friction in the drive, this was fixed by adding a bar that spanned the whole thing. The bigger bracing issue was the puncher, ~~we~~ it bent very much during the match as it was mounted by 4 stand offs. Our new ~~the~~ design mounts onto 4 bars via 10 contact points, with boxxing allaving us to hold the robot from any point.

The wedge we had worked very well pushing balls ~~robots~~ to other side.

We also found out that we have quite some time ~~when~~ before pushing triballs into goal, we would ram into bots but it was not very effective so we add a blocker that will pop up using 1 piston.

Fulbright 17/12

Fulbright was a very intresting competion, the refs made some poor calls (we should have won SF) but we were still very happy our robot worked very well.

Our puncher may not have been great but it was mostly reliable which gave us 69 points (nice) in caution skills without moving.

Our blocker was the thing that helped us win a lot of matches. We could stop very good ~~prrobe~~ shooters helping lowwer the score of the opponent.

Our elevation strategy was using the wings, this worked twice, one time our alliece thought we were stuck and pushed us off, and one time the ref said it didn't count. It was very simple and could have us a lot of points next time we just need to give the driver more practice.

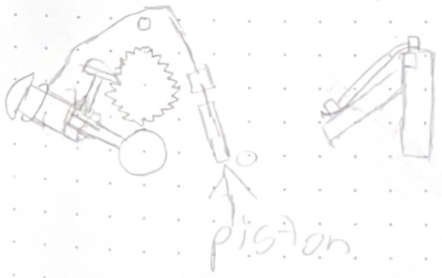
What we learnt over many comps was that there are lots of teams with good offence but not alot with good defence, therfore we will want to focus on improving blocker and wings (we will still keep a decent puncher for skills and in case our alliece cant match load.

For our next version we are making a double stage to block robots that have a high arc we will also ~~use~~ use two pistons to make the blocker less wobley

Explain string (Continued in page 13)

AWP

The way a team is ranked in qualification is by Win Point which increases with each win but also when we get the autonomous win point which is completly a series of tasks but the challeny is we need to take a ball out of the match load zone and push a ball into our own goal these two are on opposite sides which is normally done by two robots but we dont want to be reliant so the solution is shooting the ball half court, this is hard because the ball is ~~smaller~~ bigger than goal so a lot of power is needed to push in. A solo AWP mech is when we shoot with more force than the motor can pull down, where the puncher is initialy pulled down by hand. How we plan to accomplish this is we have a racket that only allows the puncher to go down but instead of a rubber band it is attached to a piston which will tention when the match starts but will release to shoot. We will mount rubberbands so that when they go up they become loose and full of puncher (still ziplined onto robot)



no tention so fall out

RGB (extends page 1

The kernel mates it so that all user progr ran on the CPU1 thred which is fast enough for most things but a neopixel needs a 800khz signal or every 1.25 microsecond taking 8 bit per color ppv pixel so 24 bits for one pixel. This results in a slow changing led and ~~slow~~ ~~complex~~ animations. James Pearman helped implement this in SPK 20220726_10_06_00 which uses CPU0 and allows for rapid comm We put a long led strip the whole way for with power injection to keep both ends the same brightness. We created a gradient by splitting the three colors, found the diffrece between two desired colors then devide by amont of leds we the increment by this amount every led. We then make a animation by putting the gradient int a vector then use std::rotate.

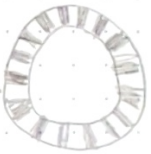
Autonomous Motion Algorythions

Autonomos movement is something that seems simple at first but it took us a long time trough hundreds of hours of testing. The goal of autonomos is moving to a desired location consitantly and accurate but the challenge is that there are many unpredictable factors like friction or being pushed.

The simplest method of movement is Dead Reackoning it has no feedback and is very unreliable. Given ow robot which travels 6 inches/second with 6 volts applied, if we want to move forward 6 inches, we apply 6 volts for 1 second. There are a lot of reasons why this wouldn't work, the most obvious are being accelaurating the robot will not reach the 6 inch

speed immedielly so will travel less than calculated. A improved verbian can take into accunt acceleration but even then, it still has many problems. A motor will not always spin the same speed with the same input voltage because of the battery level which will alter the amprage of the motor. Additionaly the friction will also differ on different floor surfaces. Therefore this algorythom is very rarely used.

The way to mesure the unpredictable factors is odometry the simplest odometry uses encoders built inside of the V5 motors. The encoders of the v5 motors act as a rotation mesure for the motor that output as square wave.



A tachometric encoder has a disk with many holes for ligh to shine trough and then optical sensors mesure the speed of oscillation

There are also two optical sensors whose pulses can be compared to get the direction by seeing which pulse goes high first. This means encoders can help us give us feedback on how much we have to spin the motor.

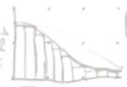
A better way of dead reakoning is to go forward until the encoder spins the desired amount but a issue is the sensor might never output 0.00 so we can set a ded band by abs(desired-error) < 0.1

```
desired = 6 //in
desired = desired/3.25*2*M_PI//rad
initial = motor.encoder(rad)
while(desired-(initial-motor.encoder(rad)
    !=0)
  drive.spin()
```

The issue is this will overshoot because when we stop the wheels will still have momentum, this can be reduced by short braking which short circuits the motor by a shunt FET across windings but this will not always stop immediety and strains the motor. We can solve this by slowing down when near the target which is called a P-controller which stands for porpotonal.

```
desired = 6 // inches
while True
    error = desired - current
    speed = error * kp
    speed = clamp(speed, -12, 12)
    drive.move(speed)
```

In a P-Controller, the speed is proportional to the remaining distance so the robot will slow down when near the desired location. A additional value called kP is a proportional gain which is a constant that we get by tuning. It decides at which distance the robot will start slowing down, if before this it will output more than the motor's max speed so we clamp the value to ±12 volts, This is a very simple controller that works well but there are many things that can still be improved. When approaching the target the error of a P controller will always be a non-zero value because the controller outputs such a low value that the robot doesn't overcome friction. We can solve this with a Integral term.



By definition a integral is the area under a curve, in this case we add all the previous error. By accumulating the integral of past errors the integral adds a correction that grows over time in response to persistent steady-state error which eliminates the steady-state error while reducing settling time.

integral ± error

Speed = (error * kP) + (integral * kI)

There are two issues to this approach when the error is zero the integral is still non-zero, so the speed is non-zero, this is mitigated by setting the integral to zero if the error is within the accepted values. Integral wind up is also a issue which is when going a far distance and when approaching the set point, the integral is invisably high so we set integral to zero every time it reaches a high value. A issue with a PI loop is oscillation and rapid unpredicted changes which is helped with a derivitive term.

```
if abs(error) <= 1
    integral = 0
if integral > huge
    integral = 0
```

Deviritive is the rate of change or gradient of a curve, We use a devitive to dampen the robots movements based on what is expected to occur. It will be a signal with the direction opposite to the direction of travel with a greater magnatude for greater speed, it typicall will be out weighed by the porpotpnal and integral components but if there is some deviation from normal the devitave will compensate. It is calculated by mesuring the diffrence between the previous error and current error. This concludes the PID controller but we still need to figure out the constats.

derivitative = error - preverro
preverror = error
power = (error * kP) + (integral * kI) + (derivitative * kD)

Tuning the controller is one of the most tedias parts of PID as it is time intensive. Here are the steps involved:

Parameter	Rise time	overshoot	settling time	steady state error	stability
kP	Decrease	Increase	N/A	Decrease	Worsen
kI	Decrease	Increase	Increase	Decrease	Worsen
kD	N/A	Decrease	Decrease	N/A	Improve

(expanded on pg 13)

1 Increase kP until slight oscillation
2 Add kD until no oscillation
3 Add kI until no steady state error

it takes a lot of tuning to get the optimal error

This PID controller was invented by engineer Nicolas Minorsky for automatic ship steering in 1922 but is now used widely for many uses in our daily life like temperature control. We learnt PID from 'A Introduction to PID' by George Gillard.

The method to implement PID is also very intresting, the simplest method as mention before is using the motor encoders, we calculate how much the wheels need to spin to move the distance using bassic geometry and then calculating the rotation by mesuring the distance between the turning center and wheels which gives the radius which can be used to calculate the arc length and rotation the motor spins. This is done by most teams on the carter MD built into each motor using the built in function but the MD is slow and all so non-tunable, so we bypass the carter and output

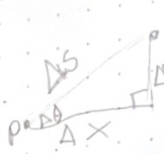
The issue with this is if we are pushed while moving the robot movement would become inaccurate. We solve this with odometry.

Robot skills would champions tlans created the concept of APS which we learnt when we met them in the VSHeat worlds. The absolute positioning system is a system which keeps tracks of the robot's absolute position in cartesian coordinates and orientation. The core of this system is odometry which converts encoder signals into a series of movements. Often tracking wheels which are rubber banded so it contacts the floor constantly to reduce wheel slippage byt the sensors which we have ordered haven't arrived yet so we use motor encoders. Normally people will have a back tracking wheel ⏚ to track horizontal but since we cant do tracking movement wheels we use a we use traction wheels so the robot cant move horizontly. Δs movement

definitions ΔL left movement
ΔR right movement
s distance tracking center to wheel
$\Delta\theta$ change in orientation
r_a middle radius
r_l left radius
r_r right radius

$\Delta L = r_L\Delta\theta$ $\Delta R = r_R\Delta\theta$ // arc length formula
$\Delta L = (r_A+s)\Delta\theta$ $\Delta R = (r_A-s)\Delta\theta$
$r_A = \frac{\Delta L}{\Delta\theta} - s$ $r_A = \frac{\Delta R}{\Delta\theta} + s$ // rearrange
$\frac{\Delta L}{\Delta\theta} - s = \frac{\Delta R}{\Delta\theta} + s$
$\Delta\theta = \frac{\Delta L - \Delta R}{2S}$ // rearrange

Using the arc length formula we can use the amount traveled by each wheel to form a arc and we can calculate the angle of that arc by plugging it to each other.



$\Delta s = \frac{\Delta L - \Delta R}{2}$

$\Delta x = \Delta s \cdot \cos(\Delta\theta)$
$\Delta y = \Delta s \cdot \sin(\Delta\theta)$

$p' = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix}$

We then can figure out the amount moved relative using basic trigo then add this to our previously known cordonates to get our current cordonate.

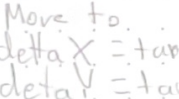
From our testing, this form of motor encoder odometry is quite accurate and drifts very little if we stay under 280 rpm, but the turning odometry was quite inaccurate, so we added a gyroscope. We then use a kalman filter which uses a prediction model to estimate probable values then will compare it with sensor values (odom and gyro) and will output the most likely value. We also use a GPS to start a match so the robot knows it's initial position, we like the gps as it is absolute positioning but has a very low refresh rate and is inaccurate while moving, how it works is it has a camera that sees the gps strips then triangulates it's own position.

The first way to use this odometry is to turn to a point with a angular PID then drive forward with a forward PID. this is what we used last year and it is very good, It is very accurate and does not drift which helped us abt with skills in the usa because their elevated field sunk in the center which ruined many people autonomous but ours worked well. For the amount we wanted to turn we used the virtual skills algorythm Alex made (max score last year) where it calculates theta with arc tan of $\frac{y}{x}$ then drives the amount calculated with pythagaryum theam. One issue is everytime we turn we have to stop which slows the robot alot.

//Turn to
deltaX = targetX - currentX
deltaY = targetY - currentY
targetTheta = M_PI - atan2(deltaX, deltaY)
angularError = std::remainder(currentTheta, - targetTheta, 2*M_PI)

//Move to
deltaX = targetX - currentX
deltaY = targetY - currentY
linearError = std::hypot(deltaX, deltaY)

Move to is very simple as it is just pythagoream theorem but turn to took a while to derive, orignally we used the function arc tangent (last year) but this took a lot of lines. The issue with atan is that it's range is $-\frac{\pi}{2} < \theta \leq \frac{\pi}{2}$ which is not the full 2π, this is due to it not reconizing opposing heading for example, since the input is $\frac{1}{1}$ and $\frac{-1}{-1}$ are the same. We had many proceding if statements to check for quardrent but shitting we found while reading documentation during fall break was atan2. This function can output $-\pi < \theta \leq \pi$ as there are two inputs instead of one so it can tell between diffrent quadrents.



One issue with turning and driving seprettly is we have to come to a complete stop everytime we switch directions so we then implemented curves. We saw a desmos graph from Purdue SIGBots with a algorgithm called a boomerang controler which we could not find any where else but it seemed like a simple algorythm so we collaborated with several other VRC to implement this. We helped implement and test the initial version. How it works is we drive towards a calculated carrot point (x_3, y_3) when we drive towards it. (using PID)

$h = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

$x_3 = x_2 - h\sin(\theta_2) \cdot d_{lead}$ ← tune curve radius

$y_3 = y_2 - h\cos(\theta_2) \cdot d_{lead}$

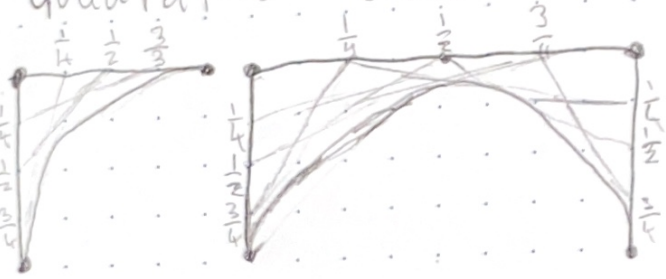
$((1-t)((1-t)x_1 + tx_3) + t((1-t)x_3 + tx_2), (1-t)((1-t)y_1 + ty_3) + t((1-t)y_3 + ty_2))$

we recalculate the point as it will change then we turn to it and repeat. When repeated at 200hz it creates a very smooth curve. A quirk we found was that this algorgithom leads the robot to follow a quadratic Bezier curve which is the same curve as Pure Pursuit. We have recenly switched to Pure Pursuit for skills as we feel the paths are more customizable and it is also faster.

Right after the fulbright scrimage, we chose to switch to Pure Pursuit after much conciderations. The benifit is that we can customize the path to however we want. Pure Pursuit is a algorithm proposed by Craig Coulter in 1992 and is commonly used by advance VRC teams. It tates a way with thousands of points which it drives following. It fixxes errorsing by drawing a circle with a radius of a constant called the look ahead distance. If set too low there will be lots of oscillation but if too high it will take a while to get back on track. It will drive towards the intersection between the circle and the line. As mentioned we switched to this because of customizable paths and also minimal tuning. One of the cool things we designed was a web app to visualize and create the path then we can download it as a array of points then import it into the code. We make the path with a combination of lines, and cubic and quadratic bezier curves. We use two curve types because they have diffrent advantages, the cubic beziers has two control points so it allows for more complex curves but takes a lot of time. The cubic bezier (similar to boomerang) is simple, and fast.

The bezier curve was created by Pierre Bézier for the body work of Renault Cars and the spline is commonly used for computer graphics and robotics as it has the ability to replicate any curve.

quadratic cubic



the basic concept is creating lines between the points to construct the curve.

We chose to use a boomerang controller for most autonomous because of PID's accuracy because we need to turn to triballs accuraly to intake while we will use Pure Pursuit for skills as we are using wings and do not need accuracy but want speed.

(continuation of pg 8) pid tuning

img 11,12

blocker continue pg 3

img 8 img 9

wings continue pg 2

img 10