

Digital Notebook

Thursday, January 19, 2023 3:29 PM

Team
40458F

Team Captain: Quinn
Drivers: Danny, Walker | Coder: Quinn
Builders: Levi , Danny, Quinn, Walker

Due to Short Notice, many pages are photos of our physical notebook, we apologize for this inconvenience, if you need help reading the photos, please feel free to ask us for help.
Many Thanks, the team members of 40458F

Table Of Contents

Thursday, January 19, 2023 7:23 PM

[Team Introduction](#)

[Game Overview](#)

[Strategy/Build Ideas](#)

[Base](#)

[Intake, Flywheel, Wheels](#)

[Base, Flywheel, Intake](#)

[Size reduction, Disk Transport Brainstorming, Flywheel](#)

[Intake, Disk Storage, Support](#)

[-Continued 1](#)

[-Continued 2](#)

[Intake Redesign + Team Changes](#)

[Disk Storage & Intake](#)

[Disk Flicker and Blue Dispenser Arm](#)

[Syracuse Comp Reflection/Overview](#)

[Code - Winter Break - 12/21/22 -- 1/03/23](#)

[Driver Skills](#)

[Full Code for Driver Control and Auton Code + Explanation](#)

Key:

[Important to driving](#)

Structural Changes

Driving Scores/Competition and general game rules

[Code](#)

Up until the [first code section](#) was written in paper notebook, we apologize for the poor quality, if you need any help understanding the writing, please feel free to talk to us

Team Introduction

Thursday, January 19, 2023 7:29 PM

Team Introductions

Team Captain: Quinn Bracken: Main Coder,
Note Booker, Builder. I am very Driven to
get to worlds, And will work to make it
Javi Harroch

Driver 1: Main Driver, ^{SE}Coder, Builder, Been on team
1yr. Likes learning electronics &

Driver 2: Danny Nuyke: Driver, Builder, Entertainer.
Been on team 1yr. Driving is fun?

HOUSSF

Been a team for 1yr, going on 2.
Went to States, Fun, but Back, Unfortunately
hadn't won any awards

9/20/22 [Z]

Game Overview

Thursday, January 19, 2023 7:31 PM

Game Overview

The object of the game is to remove and score as many disks as possible when you touch one of the contact zones over the fence. You get 1 pt per disk removed. You then shoot the disk under the fence. Depending on the area the disk lands determines additional pts. During the endgame, you can expand over the fence to contact the contact zone for additional pt per disk scored.

Corresponding

Strategy/Build Ideas

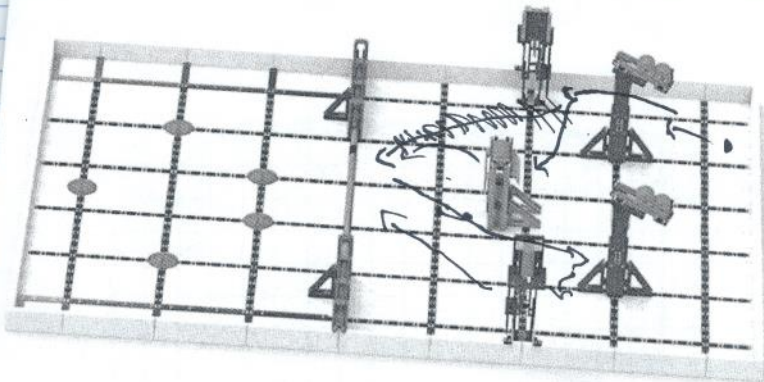
Thursday, January 19, 2023 7:32 PM

Strategy

Dump Bay Dispensers and Intake Disks, this will give us the most p/s to Speed Ratio.

~~Disperse~~ those and get Purple dispenser if we shoot
have time, Contact zones at end

1. Make Robot With Equal Dispensing Speeds
2. Fast Shooter, Flywheel?
3. Manueverable Drive.
4. Large Disk Storage



Base

Thursday, January 19, 2023 7:33 PM

Day 1 - Build a base

We are going to build a medium-base, it is more maneuverable than large

We decided to do a base close to the size limit, but not large enough that nothing can be outside the robot.

Base 2x20x34



The wheels will be in the center of the robot, to make it balanced with the supports on either side.

Days Summary:

We are 90 percent done with the base, we have the shape but not the wheels.

Problems:

Since the base was so big, we had to strengthen connectors



Connector

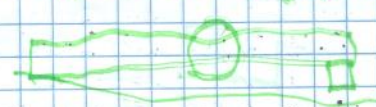
9/22/20

Intake, Flywheel, Wheels

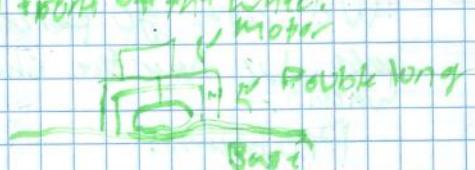
Thursday, January 19, 2023 7:34 PM

10, 11/23
9/27/22 Day 2 = Intake, flywheel, and wheels

We started by adding in the wheels. Our original plan was to put the wheels in the center with 2 TXUs on both sides to power it, because of the omni wheels smaller diameter the TXUs were lifting the entire wheels off the ground. So we decided to do a "Scotch" design, where one side of the robot scrapes the ground, with the other side being pushed up by the



after we fixed this we added on the motor mount. Since the wheels were on the inside, and the base was close to the size limit, we decided to put the motors on the inside. We achieved this by placing a double long coupler and a plate inside the base in front of the wheel.



Construction - we put intake. The intake was decided to be put on the double long side of the robot. It will be side w/ TXUs

Base, Flywheel, Intake

Thursday, January 19, 2023 7:35 PM

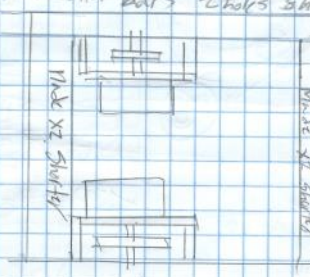
Day 3 - Base, Flywheel, Intake 10/6/22

We started off today with a plan to prototype a 1m flywheel and add on gearing for intake. Before starting that we measured our robot to make sure it was within spec.

The robot was measured to be $12\frac{1}{2}$ by $13\frac{1}{2}$ by 16 inches. The rules state that the robot's maximum size is no greater than $11 \times 9 \times 15$ (RS.6).

To make our robot's width shorter we removed the 3 beams ~~that were~~ ^{that were} widthwise, and replaced them with bars 2 holes shorter.

Intake roller was also shortened to fit thinner chain.



Back half of the robot 2/3 Davis picture!

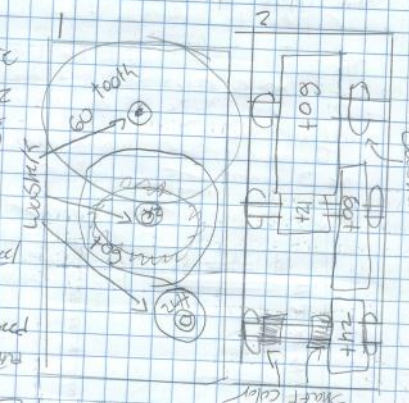
Flywheel: After prototyping a 1m flywheel with a 1250 rpm flywheel central speed ~625-630 rpm torque, removed the 2nd set of gears and 2nd motor from previous flywheel.

Continued - next page

Calculations
60/12 = 5
60/24 = 2.5
(60/24) * 2.5 = 1.5625

Day 3 - Continued 10/6/22

After removing 2nd set of gearing and motor added 2nd motor to gear-plate connections and supported gearboxes flywheel.

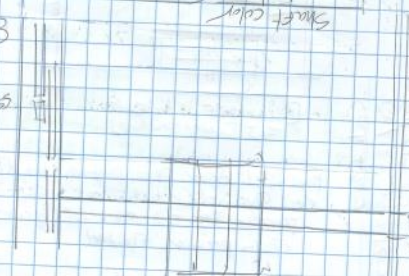


Supports are Bottom Plate Flywheel and standards (not pictured).

We also supported the other side of the flywheel with another Plate (3).

This helped speed up flywheel and freed motor for other use.

Flywheel is still under torque. May be hard across top?

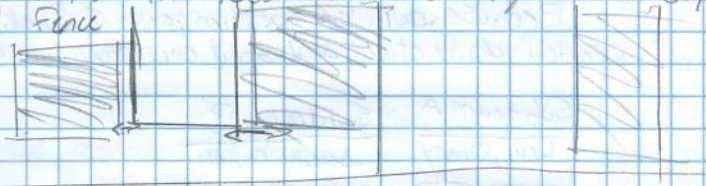


Size Reduction, Disk Transport Brainstorming, Flywheel

Thursday, January 19, 2023 7:40 PM


Day 4 - Basic Brainstorming 10/20/23

After fit, testing and driving our drivers reported that the robot had issues fitting between dispensers and fence



and turning by around them. Our fix was to remove crossbars on robot and replace w/ 2x16s

Our robot design made this easy w/ only 2 cross bars that needed to be replaced.

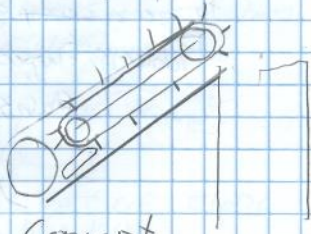


After replacing crossbars and running flywheels there was a noticeable increase in flywheel speed. After further testing the flywheel we were getting speeds of 70rpm

on the motor. Previous flywheel speed was 50rpm

Transport Ideas

- Rubber band intake transport
- Flap trap and chain transport
- Horizontal flap and chain
- Intake launches disk to flywheel



Concept

Intake, Disk Storage, Support

Thursday, January 19, 2023 7:42 PM

11/8/22
15

Intake, Storage, Support

Day 5

Because our competition is in 2 weeks, we created objectives, and split our team into Subteams

Subteam A	Subteam B
Levi, Danny	Jack, Quinn
Intake	Storage and Support

Subteam A

Because of continued issues w/ existing intake, would need Secondary rubber band wheel, which would require 2nd motor, and issues picking up disks - the top down intake was scrapped for 2 side chains

This intake was deemed by Subteam A to be a ~~unhappy~~ good mech.

Problems: Flappers are too flexible and will not pick up disks
chains will need to be in sync

-Continued 1

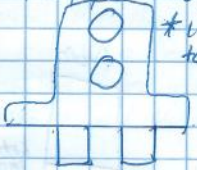
Thursday, January 19, 2023 7:43 PM

11/8/22
16

Intake Storage Support Day 5 - continued

Solutions: replace rubber flappers w/
rigid structural pieces

To make chains in sync
we attached the top 2
gears onto a plank
with 2 gears and a chain on top
gears dischain to link
(reverse of) two gears



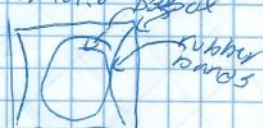
*width is
too wide

Subteam b

Storage:

Objectives: working storage
container that holds
10+ disks
Container must not let
disks come out bottom
while driving

Ideas: box that intake drops disks into
rubber bands along bottom
for friction

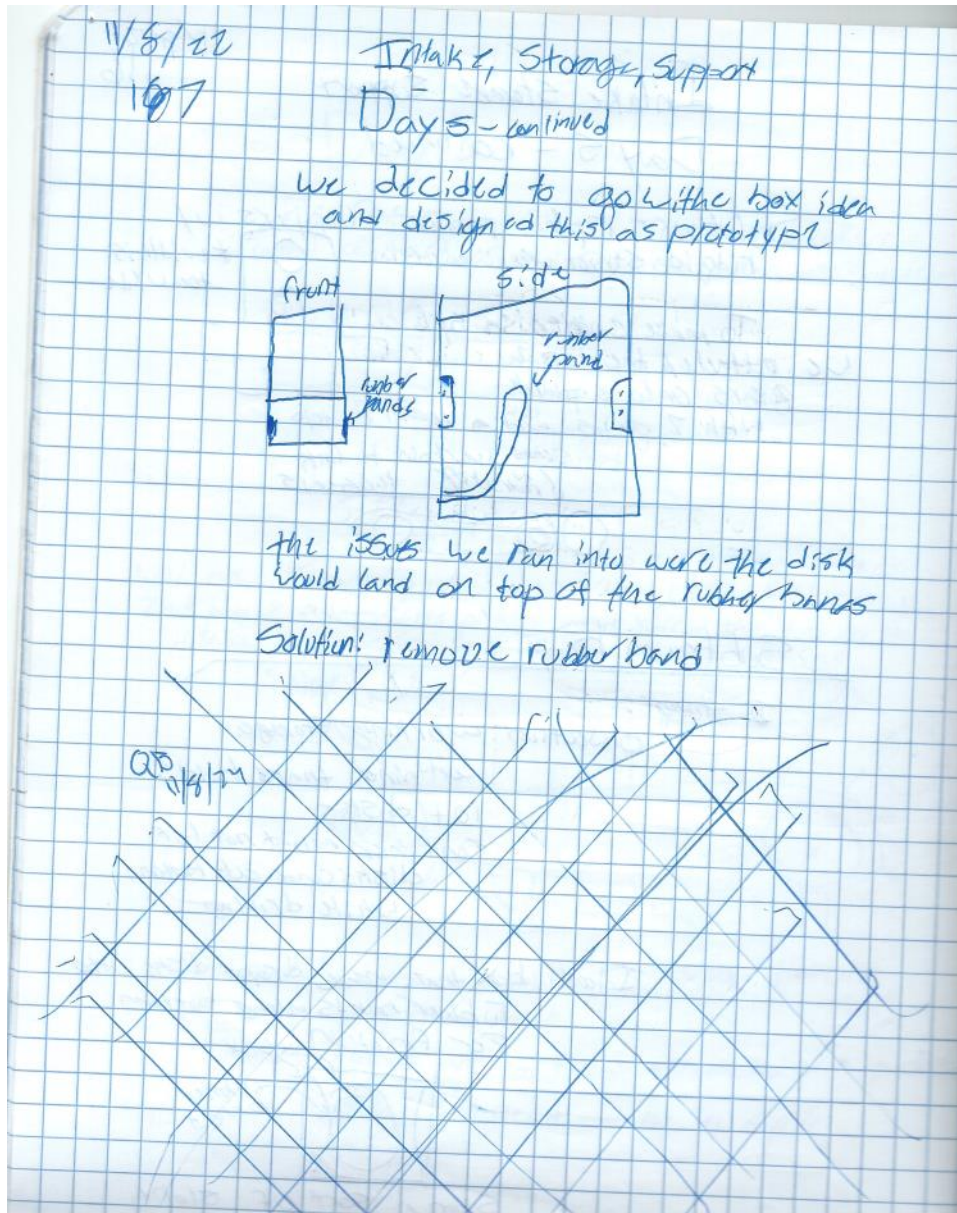


Some sort of slope
to hold disks

-Continued 2

Thursday, January 19, 2023

7:44 PM



Intake Redesign + Team Changes

Thursday, January 19, 2023 7:45 PM

Intake Redesign and Team Changes

11/10/22
18

* One of our teammates is moving at the end of the week, so we are taking on 1 new team member. Jack Nydegger had 1 year of Robotics experience 2 yrs ago. We also had a late joiner, Justin Roberts, and because our team had the least people, he joined us. Walker, no experience.

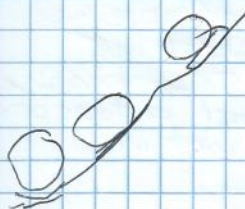
Intake Redesign

Sideways flapper intake is not viable due to the shortage of time to comp.

So, we are trying a top-down roller design.

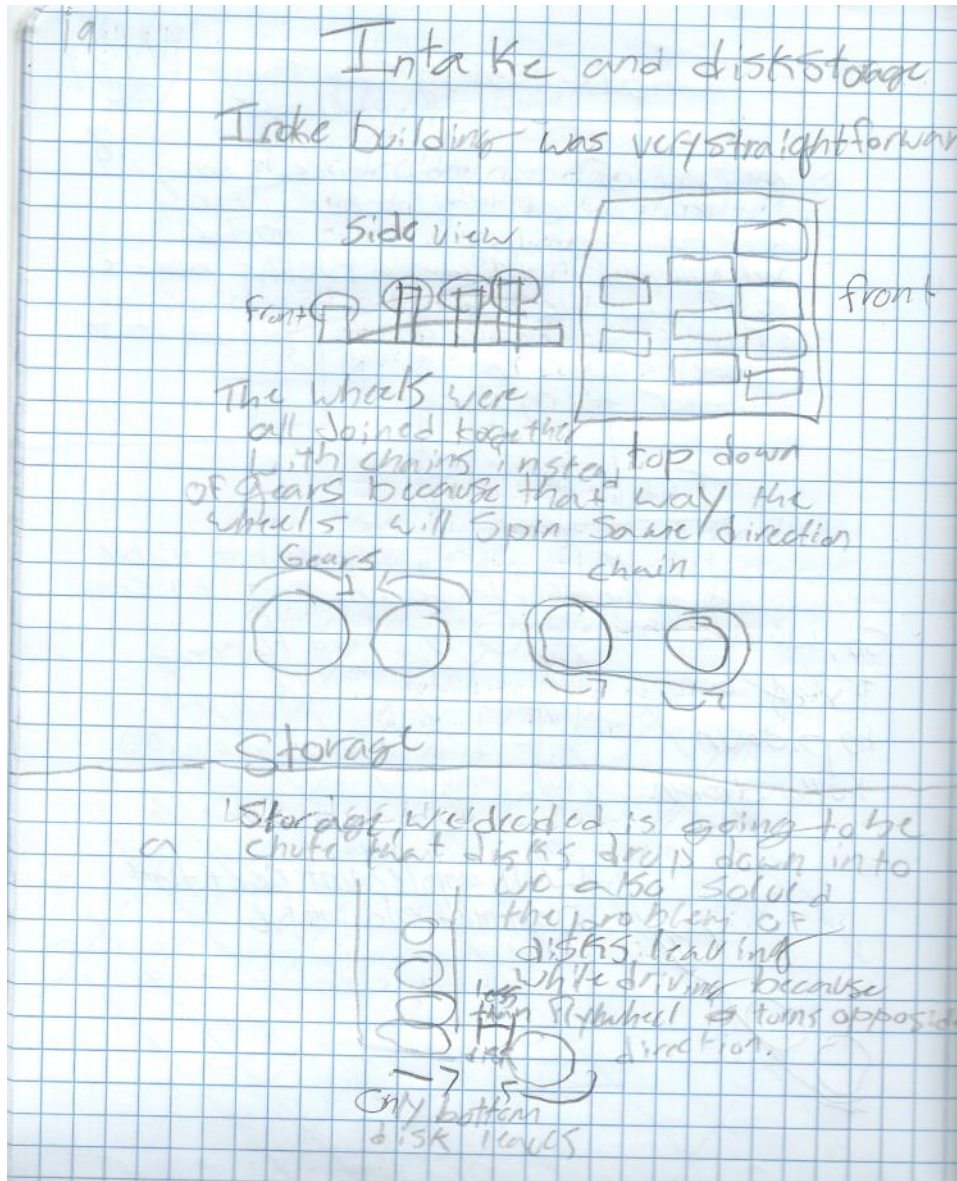


Will implement next time
Dismantled old intake



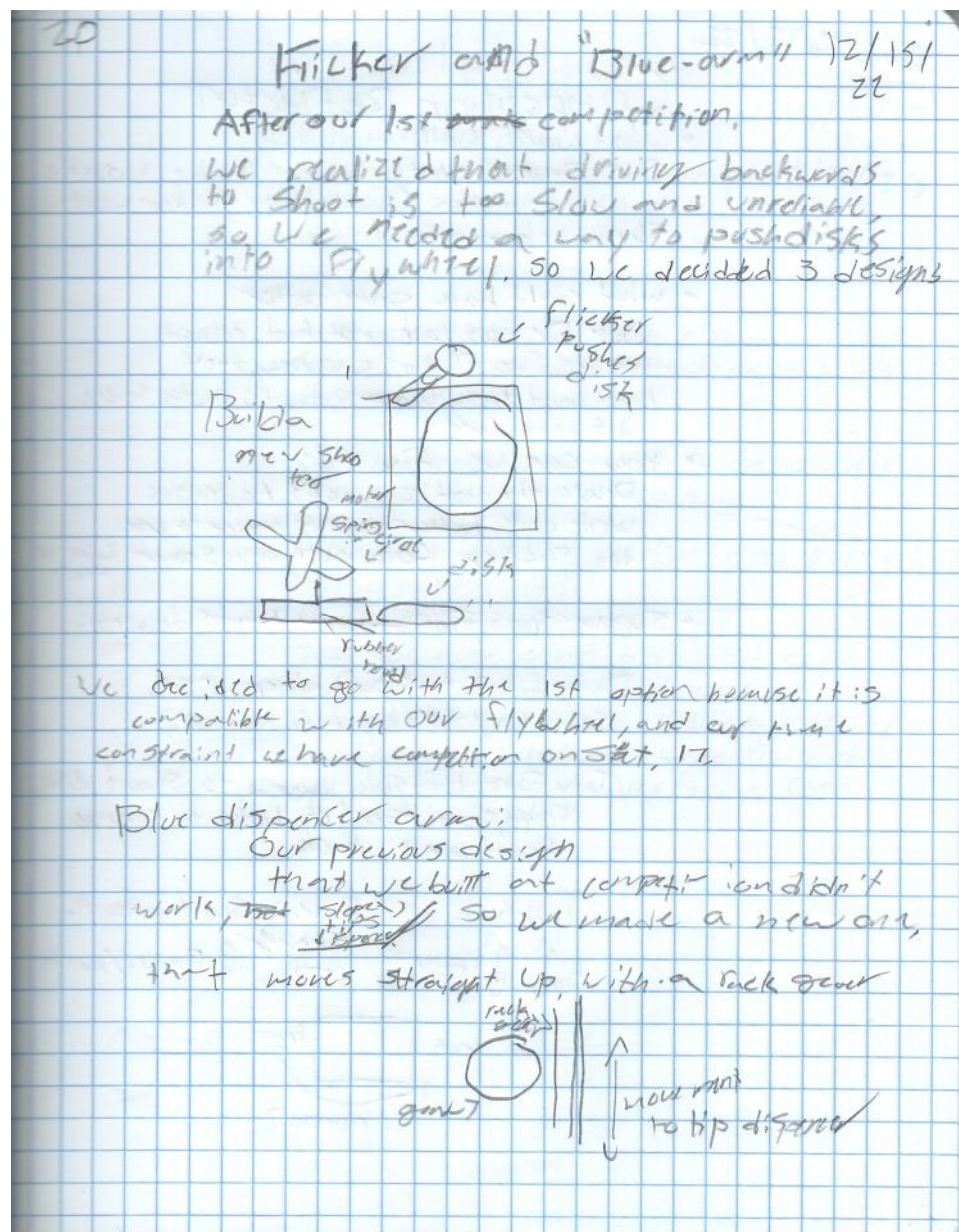
Disk Storage & Intake

Thursday, January 19, 2023 7:46 PM



Disk Flicker & Blue Dispenser Arm

Thursday, January 19, 2023 7:47 PM



Syracuse Comp Reflection/Overview

Thursday, January 19, 2023 7:48 PM

24 12/17/22

Competition Reflection

- What went well:
Our Flywheel shot well, Our intake was able to pick up disks. Quickly built flicker was storage held enough disks
- What could have gone better:
Our Drivers have not had enough practice, since flicker was new, they had hard time using it. no auto, severe dis advantage
- How can we fix it?:
Devote the next 2 weeks to practice. Write code better so it is easier to use the flicker. Code autonomous over winter
- Suggestions from previous years drivers:
Denny likes tank, but walker RSA, some way to stop mid match?
They found backside hard to shoot disk from, some sort of button to reverse front and back
Tank
Motor
Reverse Split arcade
Turning
Forward/back
back front
reverse?

Code - Winter Break - 12/21/22 -- 1/03/23

Thursday, January 19, 2023 7:49 PM

Overview:

As Team Captain I was tasked with bringing home the robot, and keeping it over Winter Break 12/21/22 - 1/03/23, I judged that it would be wisest to work on Driver Code and implement some of the changes that the drivers had requested. Some of the Changes requested included:

- Separate Driver Controls, each of the drivers preferred different driving schemes
- Ease of use for Intake, Flywheel, and Blue Arm Controls
- Reversible Driver Controls, so it would be easier to aim disks

Ideas/Initial Thoughts:

- Find out what driver controls the drivers wanted
- Custom Driver Control would mean removing drivetrain from code config
- If I had time, maybe a way to copy driver actions for autonomous?

Implementation:

After Looking at the driver control #pragma code (this is where the code that vexcode automatically generates is stored) I discovered a couple things, Arcade controls, the driving style one of the drivers preferred, was simpler than I imagined, and Vexcode automatically creates motor groups with the motors for each side of the drivetrain, this means that I do not have to remove the drivetrain from the vexcode devices config. The arcade control boils down to just Power Axis(the axis which just moves the robot forward or back) - or +(depending on which side) Turn Axis (the axis on the joystick that turns the robot). To start, I thought I could just modify the #pragma driver code, which already includes deadbands for joystick drift, start with the default Arcade Controls, and add in a Boolean value (true or false) to decide which control scheme to use, so the code looked something like this:

```
If (tank) {  
    Leftside = LeftJoystickYAxis  
    Rightside = RightJoystickYAxis  
} else {  
    Leftside = LeftJoystickYAxis + RightJoystickXAxis  
    Rightside = LeftJoystickYAxis - RightJoystickXAxis  
}
```

This says, if tank controls are enabled, set the values for tank, otherwise use arcade

After talking with the driver who wanted arcade, I misunderstood, he wants the Right Joystick to control power, and left to control turning. I modified the code so that this was now the case:

```
If (tank) {  
  Leftside = LeftJoystickYAxis  
  Rightside = RightJoystickYAxis  
} else {  
  Leftside = RightJoystickYAxis + LeftJoystickXAxis  
  Rightside = RightJoystickYAxis - LeftJoystickXAxis  
}
```

This code now drives like the driver requested it, however after saving, I realized something, the #pragma code resets after each close and reopen of the program. To fix this I copied the entire driver control portion of the #pragma code, and its variables, and put it in my main code. After uploading, the driver control now didn't work, it acted like the default arcade controls. I realized that the code I made conflicts with the preset driver code, so I removed the drivetrain controls from the controller configuration menu. This fixed the problem

The drivers requested this control set for the buttons:

Top Right Shoulder button Controls Intake Spinning or not, the bottom one stops intake

E Up and Down Control the Blue Arm

F Up starts and controls different modes of the flywheel

F Down Stops Flywheel

The Blue Arm controls were easily set up in the device config menu

F Up and down were just setting event handlers one for starting and changing modes, and one for stopping

Changing modes was accomplished with another boolean value

Shoulder buttons were set with another "mode" boolean value, one for holding, to spin intake backwards, and one to spin forwards, and an event handler for stopping

Problems:

Because of time constraints, I was not able to implement reversible driver controls, though in theory it should be very quick.

Drivers have a hard time adjusting to new control set.

Solutions:

Drivers Practice adjusting to the new control

Allocate some time to working on code, earliest possible after next competition?

Driver Skills

Thursday, January 19, 2023 8:23 PM

As our robot was almost finished, we turned our attention to practicing, these are our scores for matches, we wanted to get at least 15 done, but due to time constraints we were only able to get through 8

Looking at World and State standings, we are currently in 26 in the state, with a 48 in driver and a 10 in autonomous. If the autonomous code could be improved to be more accurate, and possibly code it to shoot, that would mean a 29-50 pt autonomous(variability due to auton accuracy) and up to a 65 pt driver skills, accounting for nervousness and other factors during comp.

This would total to a $45+60 = 105$ pt skills total which would put us at 8th as of 1/20/23 this will change though the season though

Driver Scores 1/19/23

55	37	42	45	43	47
77	48				

Code for Autonomous, and Full Code for Driver

Thursday, January 19, 2023 8:26 PM

Autonomous code was just making the robot drive forward and turn over and over, and isn't worth explaining in detail. The general Strategy was to get the blue dispensers, the yellow and then shoot. However, due to time constraints, it was not coded to shoot, yielding in a 29 or sometimes 31 pt auto, because a disk would fall from the blue dispenser, and under the bar.

```
#pragma region VEXcode Generated Robot Configuration
// Make sure all required headers are included.
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <string.h>

#include "vex.h"
using namespace vex;
// Brain should be defined by default
brain Brain;

// START IQ MACROS
#define waitUntil(condition) \
do { \
    wait(5, msec); \
} while (!(condition)) \
#define repeat(iterations) \
for (int iterator = 0; iterator < iterations; iterator++) \
// END IQ MACROS

// Robot configuration code.
inertial BrainInertial = inertial();
motor LeftDriveSmart = motor(PORT3, 1, false);
motor RightDriveSmart = motor(PORT4, 1, true);
gyro DrivetrainGyro = gyro(PORT10, true);
smartdrive Drivetrain = smartdrive(LeftDriveSmart, RightDriveSmart,
DrivetrainGyro, 200);
motor Bluearm = motor(PORT8, true);
motor Intake = motor(PORT1, true);
motor Flywheel = motor(PORT7, false);
void calibrateDrivetrain() {
    wait(200, msec);
    Brain.Screen.print("Calibrating");
    Brain.Screen.newLine();
    Brain.Screen.print("Gyro");
    DrivetrainGyro.calibrate(calNormal);
    while (DrivetrainGyro.isCalibrating()) {
        wait(25, msec);
    }
}
```

```

    }
    // Clears the screen and returns the cursor to row 1, column 1.
    Brain.Screen.clearScreen();
    Brain.Screen.setCursor(1, 1);
}
#pragma endregion VEXcode Generated Robot Configuration
// Include the IQ Library
#include "vex.h"

// Allows for easier use of the VEX Library
using namespace vex;
float myVariable;
// "when started" hat block
int whenStarted1() {
    Drivetrain.driveFor(reverse, 20.0, mm);
    Drivetrain.turnToRotation(44.0, degrees);
    Drivetrain.driveFor(reverse, 65.0, mm);
    Drivetrain.turnToRotation(55.0, degrees);
    Bluearm.spinToPosition(285.0, degrees);
    wait(0.25, seconds);
    Bluearm.spinToPosition(0.0, degrees);
    Drivetrain.turnToHeading(90.0, degrees);
    Drivetrain.driveFor(forward, 200.0, mm);
    Drivetrain.turnToHeading(-102.0, degrees);
    Drivetrain.driveFor(reverse, 200.0, mm);
    Bluearm.spinToPosition(300.0, degrees);
    wait(0.25, seconds);
    Bluearm.spinToPosition(0.0, degrees);
    Intake.setVelocity(100.0, percent);
    Flywheel.setVelocity(40.0, percent);
    Flywheel.spin(reverse);
    Intake.spin(forward);
    Drivetrain.turnToHeading(180.0, degrees);
    Drivetrain.setHeading(0.0, degrees);
    Drivetrain.turnToHeading(-5.0, degrees);
    Drivetrain.setHeading(0.0, degrees);
    Drivetrain.driveFor(forward, 200.0, mm);
    Drivetrain.turnToHeading(-20.0, degrees);
    Drivetrain.driveFor(forward, 750.0, mm);
    Drivetrain.turnToHeading(-90.0, degrees);
    Drivetrain.driveFor(reverse, 350.0, mm);
    Bluearm.spinToPosition(90.0, degrees);
    Drivetrain.setTurnVelocity(100.0, percent);
    Drivetrain.turnToHeading(-60.0, degrees);
    //New Stuff
    wait(1, seconds);
    Drivetrain.turnToHeading(-90, degrees);
    Drivetrain.driveFor(forward, 50, mm);

    return 0;
}

int main() {
    // Calibrate the Drivetrain Gyro
    calibrateDrivetrain();
    whenStarted1();
}

```

```
}
```

And the Driver Code:

```
#pragma region VEXcode Generated Robot Configuration
// Make sure all required headers are included.
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <string.h>

#include "vex.h"
using namespace vex;
// Brain should be defined by default
brain Brain;

// START IQ MACROS
#define waitUntil(condition) \
do { \
    wait(5, msec); \
} while (!(condition)) \
#define repeat(iterations) \
for (int iterator = 0; iterator < iterations; iterator++) \
// END IQ MACROS

// Robot configuration code.
inertial BrainInertial = inertial();
motor LeftDriveSmart = motor(PORT3, 1, false);
motor RightDriveSmart = motor(PORT4, 1, true);
drivetrain Drivetrain = drivetrain(LeftDriveSmart, RightDriveSmart, 200, 173, 76,
mm, 1);
controller Controller = controller();
motor BlueArm = motor(PORT8, true);
motor Intake = motor(PORT1, true);
touchled Ligh = touchled(PORT12);
motor Flicker = motor(PORT9, true);
motor Flywheel = motor(PORT7, false);

// define variable for remote controller enable/disable
bool RemoteControlCodeEnabled = true;
// define variables used for controlling motors based on controller inputs
bool eButtonsControlMotorsStopped = true;
// define a task that will handle monitoring inputs from Controller
int rc_auto_loop_function_Controller() {
    // process the controller input every 20 milliseconds
    // update the motors based on the input values
    while(true) {
        if(RemoteControlCodeEnabled) {
            // check the ButtonEUp/ButtonEDown status to control BlueArm
            if (Controller.ButtonEUp.pressing()) {
                BlueArm.spin(forward);
                eButtonsControlMotorsStopped = false;
            } else if (Controller.ButtonEDown.pressing()) {
```



```

        BlueArm.spin(reverse);
        eButtonsControlMotorsStopped = false;
    } else if (!eButtonsControlMotorsStopped) {
        BlueArm.stop();
        // set the toggle so that we don't constantly tell the motor to stop when
the buttons are released
        eButtonsControlMotorsStopped = true;
    }
}
// wait before repeating the process
wait(20, msec);
}
return 0;
}
task rc_auto_loop_task_Controller(rc_auto_loop_function_Controller);
#pragma endregion VEXcode Generated Robot Configuration
//Declare
// Include the IQ Library
#include "vex.h"

//Stuff for driver control
bool DrivetrainNeedsToBeStopped_Controller;
bool DrivetrainRNeedsToBeStopped_Controller;
//Fup Counter for flywheel
float FUPCounter = 0;
//I don't think this is needed, but I'll leave it just in case I forgot about
something
bool Stop;
//For flicker
bool x = true;
//Spinner direction 0 is stopped, 1 is reverse 40%, and 2 is forward 100%
float spinner = 0;
//For Driver control, this is to reverse the controls
bool reversed = false;

// Allows for easier use of the VEX Library
using namespace vex;
//Brain and Console stuff for printing see the switch function
int Brain_precision = 0, Console_precision = 3;
//Precision delay for motors and writing to Sd card
int precision = 100;
//Weather danny is driving or walker; danny is tank, walker is reversed split
arcade
//Also drivetrain stuff
int Danny = false, drivetrainLeftSideSpeed, drivetrainRightSideSpeed;
//Float values for SD card
float xaccel, rot, wheelturnsl, wheelturnsr, bluearmturns, intaketurns,
flicker_max = 18, flicker_min = 0, flickerpos, flywheelPos;
// Used to find the format string for printing numbers with the
// desired number of decimal places
const char* printToConsole_numberFormat() {
    // look at the current precision setting to find the format string
    switch(Console_precision){
        case 0: return "%.0f"; // 0 decimal places (1)
        case 1: return "%.1f"; // 1 decimal place (0.1)
        case 2: return "%.2f"; // 2 decimal places (0.01)
        case 3: return "%.3f"; // 3 decimal places (0.001)
    }
}

```

```

        default: return "%f"; // use the print system default for everthing else
    }
}

//Code For the Driver Control, I took this from the stuff vexcode automatically
makes, and modded it
int driving(){
    while(true) {
        if (Danny) {
            if (reversed) {
                drivetrainLeftSideSpeed = -(Controller.AxisA.position());
                drivetrainRightSideSpeed = -(Controller.AxisD.position());
            }else {
                drivetrainLeftSideSpeed = Controller.AxisA.position();
                drivetrainRightSideSpeed = Controller.AxisD.position();
            }
        } else {
            if (reversed) {
                drivetrainLeftSideSpeed = -(Controller.AxisD.position() -
Controller.AxisB.position());
                drivetrainRightSideSpeed = -(Controller.AxisD.position() +
Controller.AxisB.position());
            }else {
                drivetrainLeftSideSpeed = Controller.AxisD.position() +
Controller.AxisB.position();
                drivetrainRightSideSpeed = Controller.AxisD.position() -
Controller.AxisB.position();
            }
        }
        if (drivetrainLeftSideSpeed < 5 && drivetrainLeftSideSpeed > -5) {
            // check if the left motor has already been stopped
            if (DrivetrainLNeedsToBeStopped_Controller) {
                // stop the left drive motor
                LeftDriveSmart.stop();
                // tell the code that the left motor has been stopped
                DrivetrainLNeedsToBeStopped_Controller = false;
            }
        } else {
            // reset the toggle so that the deadband code knows to stop the left
motor nexttime the input is in the deadband range
            DrivetrainLNeedsToBeStopped_Controller = true;
        }
        // check if the value is inside of the deadband range
        if (drivetrainRightSideSpeed < 5 && drivetrainRightSideSpeed > -5) {
            // check if the right motor has already been stopped
            if (DrivetrainRNeedsToBeStopped_Controller) {
                // stop the right drive motor
                RightDriveSmart.stop();
                // tell the code that the right motor has been stopped
                DrivetrainRNeedsToBeStopped_Controller = false;
            }
        } else {
            // reset the toggle so that the deadband code knows to stop the right
motor next time the input is in the deadband range
            DrivetrainRNeedsToBeStopped_Controller = true;
        }
    }

    // only tell the left drive motor to spin if the values are not in the

```

```

deadband range
    if (DrivetrainLNeedsToBeStopped_Controller) {
        LeftDriveSmart.setVelocity(drivetrainLeftSideSpeed, percent);
        LeftDriveSmart.spin(forward);
    }
    // only tell the right drive motor to spin if the values are not in the
deadband range
    if (DrivetrainRNeedsToBeStopped_Controller) {
        RightDriveSmart.setVelocity(drivetrainRightSideSpeed, percent);
        RightDriveSmart.spin(forward);
    }

    // wait before repeating the process
    wait(20, msec);
}
}
//End Code For Driver

```

```

// Write To SD Card and Console
int whenStarted1() {
    //Calibrate Inertial
    BrainInertial.calibrate();
    while (BrainInertial.isCalibrating()) { task::sleep(50); }

    while (true) {

        //Set Data
        xaccel = BrainInertial.acceleration(xaxis)*986; //Reports in mm
        rot = BrainInertial.rotation(degrees);
        wheelturnsl = LeftDriveSmart.position(turns);
        wheelturnsr = RightDriveSmart.position(turns);
        bluearmturns = BlueArm.position(turns);
        intake turns = Intake.position(turns);
        flickerpos = Flicker.position(turns);
        flywheelPos = Flywheel.position(turns);

        //Print to console stuff, disabled for speed
        //printf(printToConsole_numberFormat(), static_cast<float>(xaccel));
        //Print Data to Console

        //Print Left Wheel Rotations
        printf(printToConsole_numberFormat(), static_cast<float>(wheelturnsl));
        printf(",");
        //Print Right Wheel Rotations
        printf(printToConsole_numberFormat(), static_cast<float>(wheelturnsr));
        //New Line
        printf("\n");

        //Open AutoData file in append mode
        FILE* AutoData = fopen("AutoData.txt", "a");
        Ligh.setColor(white);

        //Print data to SD card
    }
}

```



```

//Rotation
//fprintf(AutoData, "%8.2f", rot);
//New Line
//fprintf(AutoData, ",");
//Left Wheel Turns
//float time = Brain.Timer.value();
//fprintf(AutoData, "%2.2f\n", time);
fprintf(AutoData, "%2.2f", wheelturnsl);

//New Line
fprintf(AutoData, "\n");
//Right Wheel Turns
fprintf(AutoData, "%2.2f", wheelturnsr);
fprintf(AutoData, "\n");
//Blue Arm Turns
fprintf(AutoData, "%2.2f", bluearmturns);
fprintf(AutoData, "\n");
//Flicker
fprintf(AutoData, "%2.2f", flickerpos);
fprintf(AutoData, "\n");

//IntakeTurns
fprintf(AutoData, "%2.2f", intaketurns);
fprintf(AutoData, "\n");

//Flywheel
fprintf(AutoData, "%2.2f", flywheelPos);

//New Line

fprintf(AutoData, "\n");

/* I tried this, but it is more complicated
char test[256];
sprintf(test, "%8.2f", wheelturnsl);
fputs(test, AutoData);
fputs(",", AutoData);
sprintf(test, "%8.2f", wheelturnsr);
fputs(test, AutoData);
*/
//Close File
fclose(AutoData);

//Wait for the determined interval 0.1 sec
//This can be changed to suit responsiveness of robot
wait(precision, msec);
}
return 0;
}
//End SD Card Stuff

```

```

// ShoesStuff
int whenStarted2() {
    Drivetrain.setDriveVelocity(100,percent);
    Drivetrain.setTurnVelocity(80,percent);
    BlueArm.setVelocity(100,percent);
    Intake.setVelocity(100,percent);
    Flicker.setVelocity(100,percent);
    BlueArm.setStopping(hold);
    // Set F Up ounter to 0 to start
    FUPCounter = 1;

    Ligh.setColor(red);
    return 0;
    while(true) {

        //Brain.Screen.print(motpos);

        Brain.Screen.clearScreen();
    }//closed
} //closed
// "when Controller ButtonFUp pressed" hat block
void onevent_ControllerButtonFUp_pressed_0() {
    if (FUPCounter == 1.0) {
        Ligh.setColor(blue);
        // If Button Toggled On And Pressed
        // Start Slowing
        Flywheel.setVelocity(50.0, percent);
        Flywheel.spin(forward);
        wait(0.2, seconds);
        Flywheel.setVelocity(20.0, percent);
        Flywheel.spin(forward);
        wait(0.2, seconds);
        // Reverse Speed
        Flywheel.setVelocity(40.0, percent);
        Flywheel.spin(reverse);
        // Reset Toggle (Change var to Bool)
        FUPCounter = 0.0;
        //Set Data
        spinner = 1;
    } //if closed
    else {
        // Otherwise
        // When Button Pressed Slow flywheel, and set right direction, then speed up
        Ligh.setColor(green);
        Flywheel.setVelocity(20.0, percent);
        Flywheel.spin(reverse);
        wait(0.2, seconds);
        Flywheel.setVelocity(100.0, percent);
        Flywheel.spin(forward);
        // Set Toggle (Change var to Bool)
        FUPCounter = 1.0;
        //Data
        spinner = 2;
    } //elseclosed
}

```

```

} //closed
// "when Controller ButtonRUp pressed" hat block
void onevent_ControllerButtonRUp_pressed_0() {
    // When Button Pressed, Set Intake reverse (only for jams)
    printf("LUP");
    Intake.setVelocity(80.0, percent);
    Intake.spin(reverse);
    //onevent_ControllerButtonFUp_pressed_0();
} //closed
// "when Controller ButtonRUp released" hat block
void onevent_ControllerButtonRUp_released_0() {
    // When Button Released Reset Intake (Also start button)
    Intake.setVelocity(100.0, percent);
    Intake.spin(forward);
}
} //closed
// "when Controller ButtonRDown pressed" hat block
void onevent_ControllerButtonRDown_pressed_0() {
    Intake.stop();
} //closed
// "when Controller ButtonFDown pressed" hat block
void onevent_ControllerButtonFDown_pressed_0() {
    Ligh.setColor(red);
    Flywheel.stop();
    //Set Data
    spinner = 0;
} //closed
void onevent_ControllerButtonLDown_Pressed_0() {
    x = true;
    BlueArm.spinFor(forward, 90, degrees);
    while (x) {
        //Flicker.spinToPosition(flicker_max, degrees);
        //Flicker.spinToPosition(flicker_min, degrees);
        Flicker.spin(forward);

    } //closed
    Flicker.stop();
} //closed
void onevent_controllerButtonLDown_Released_0(){
    x = false;
    //closed
}
void onevent_controllerButtonEUp_Pressed_0(){

    //closed

    //closed
}
void extras() {
    wait(1, seconds);
    BlueArm.setStopping(hold);

    Flywheel.setVelocity(100, percent);

    //Closed

}
void onevent_controllerButtonL3_Pressed_0() {

```



```

    if (Danny) {
        Danny = false;
    } else {
        Danny = true;
    }
}
void onevent_controllerButtonR3_Pressed_0() {
    if (reversed) {
        reversed = false;
    } else {
        reversed = true;
    }
}

int main() {
    // Calibrate the Drivetrain Gyro
    //calibrateDrivetrain();
    //Set Maxes
    BlueArm.setMaxTorque(100,percent);
    Flywheel.setMaxTorque(100,percent);
    Flywheel.setStopping(coast);
    Flicker.setMaxTorque(100,percent);

    //FUP
    Controller.ButtonFUp.pressed(oneevent_ControllerButtonFUp_pressed_0);
    //From Shoes, Event Handlers
    //Intake Stuff
    //RUP
    Controller.ButtonRUp.pressed(oneevent_ControllerButtonRUp_pressed_0);
    Controller.ButtonRUp.released(oneevent_ControllerButtonRUp_released_0);
    //RDown
    Controller.ButtonRDown.pressed(oneevent_ControllerButtonRDown_pressed_0);
    //Flywheel Stop
    Controller.ButtonFDown.pressed(oneevent_ControllerButtonFDown_pressed_0);
    //LDown
    Controller.ButtonLDown.pressed(oneevent_ControllerButtonLDown_Pressed_0);
    Controller.ButtonLDown.released(oneevent_controllerButtonLDown_Released_0);
    //BlueArm
    Controller.ButtonEUp.pressed(oneevent_controllerButtonEUp_Pressed_0);
    //Controller.ButtonEDown.pressed(oneevent_controllerButtonEDown_Pressed_0);
    Controller.ButtonEUp.released(oneevent_controllerButtonEUp_Pressed_0);
    //Controller.ButtonEDown.released(oneevent_controllerButtonEDown_Pressed_0);
    Controller.ButtonL3.pressed(oneevent_controllerButtonL3_Pressed_0);
    Controller.ButtonR3.pressed(oneevent_controllerButtonR3_Pressed_0);
    //Driving
    vex::task ws1(driving);
    //ShoesInit
    //vex::task ws2(extras);
    //Shoes Stuff
    vex::task ws3(whenStarted2);

    //SD Card Suff
    //whenStarted1();
}

```