THE EYES OF THE ROBOT

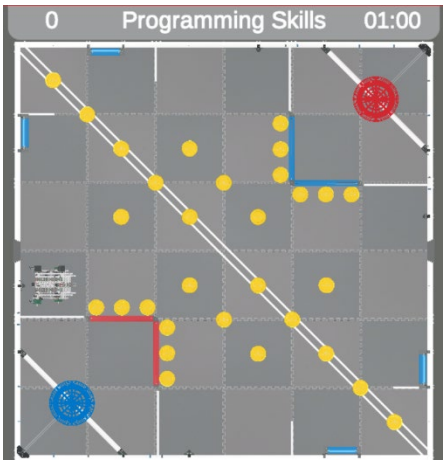
VRC SPIN UP 2022-2023

PARTICIPATING MEMBERS: REMIEL LIN

TABLE OF CONTENTS

1. **Introduction:**
   a. **VR Skills**
   b. **The Robot**
   c. **Code Format**
2. **About the Submitted Run**
3. **Functions**
   a. **The Use of Variables in Functions**
4. **Main Program**

INTRODUCTION

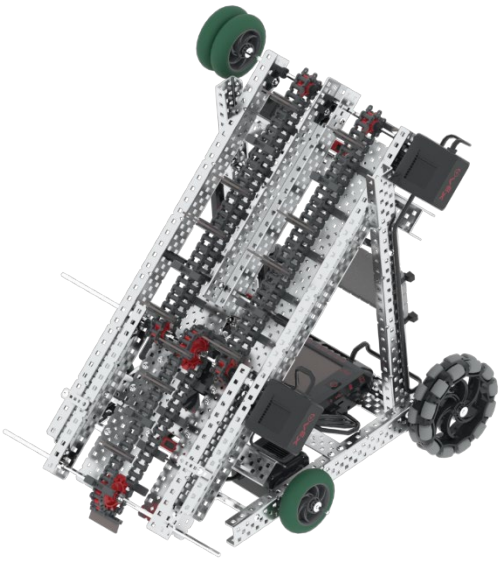


VR Skills

Imagine a world, where the field gets reset instantly and all parts of the robot function without error. Did you guess virtual skills? Not quite, but the VR skills environment allows for programmers to hone their skills in coding, as well as having a better understanding of programming! For me, this meant trying out sensors that I never had the opportunity to use and learning Python, the language VR skills is set in. Before VR skills I never had a chance to use conditionals my code, nor did I have a sophisticated understanding of it. But after I started using VR skills, it has taught me a very important aspect of Coding. Math. From all of those equations I have passed as "completely useless" ended up playing a crucial role in coding functions such as driving to a coordinate on the field. By the completion of my Skills program, I learned how to make functions with a sophisticated understanding of sensors, conditionals, and math.

Disco: The Robot used in VR Skills

Disco is equipped with 5 Sensors: The inertial, distance, line, optical, and Game Positioning Sensors. All of which are used in the program. The sensors are used to determine when tasks in the game such as turning a roller or intaking a disc are completed, so that the robot may move as efficiently as possible. Sensors such as the inertial or GPS sensor are also used to mitigate errors using information that they collect about the robot's position and orientation.



Code Format: (FINISH THIS LATER)

Spacing: In the main code, large chunks of space indicate a new "section" whereas intaking a set of discs could be one section and shooting them can be another

ABOUT THE RUN

Points Breakdown:

- 292 Points Total
- 48 discs scored in high goals
    - 5 points each disc
- All 4 rollers turned
    - 10 points per roller
- 4 tile expansion
    - 3 points per tile covered

Robot path

FUNCTIONS

Turn Roller Function

Sensors Used: Optical Sensor

Programming structures: While loops

Purpose of Function:

The purpose of the Turn Roller Function is to allow for the robot to detect when the roller has been turned to the red side. The robot spins it's intake (roller attached) until this is the case.

```
40  #function to turn roller
41  #uses the optical sensor and while loop conditionals
42  def turnroller():
43      intake_motor_group.set_velocity(80,PERCENT)
44      intake_motor_group.spin(REVERSE)
45      while not roller_optical.color() == RED.value:
46          wait(1, MSEC)
47      #detects weather or not the roller is red
48      while not roller_optical.color() == BLUE.value:
49          wait(1, MSEC)
50      #detects weather or not the roller is blue
51      intake_motor_group.stop()
52      #if the optical sensor detects that the roller has been turned to the red side, the intake stops
```

Shoot Disc Function

Sensors Used: Line Tracker Sensor

Programming structures: While loops, For loops

Purpose of Function:

The shoot disc function takes in a variable: the number of discs it needs to shoot. The function counts the number of discs passing through the top line tracker. Every disc that passes through it gets fired by Disco (name of robot). The for loop matches the number of discs you want the bot to shoot with the number of discs shot. If the values are equal, the function is completed. Otherwise the bot continues moving it's intake reverse until the disc is shot out

```
57  #variables used in the shoot disc function
58  discstoshoot = 0
59  #this variable is assigned every time the function is called, it tells the function how many discs it needs to shoot
60
61  #function to fire the disc
62  #uses the line sensor, while loops, and for loop conditionals
63  def shootdisc(discstoshoot):
64      intake_motor_group.spin(REVERSE)
65      for x in range(discstoshoot):
66      #for loop repeats this sequence depending on the numerical value assigned in the "discstoshoot" variable
67          while not top_line_tracker.reflectivity(PERCENT) > 80:
68              wait(5, MSEC)
69          #This conditional detects weather or not the disc is at the top of the intake using the line tracker sensor
70          #If it is, then the program moves on to the next line
71          while not top_line_tracker.reflectivity(PERCENT) < 20:
72              wait(5, MSEC)
73          #This conditional allows for the disc to fire: the intake will spin until the sensor is no longer detecting the disc
74          #after the sensor detects that the disc is no longer there,
75          #it will acknowldge that it has shot 1/discstoshoot and will repeat until it has shot all of the discs
```

Drive to Disc Function

Sensors Used: Distance Sensor

Programming structures:
While loops

Purpose of Function:

At the front foot of the robot lies a distance sensor pointed at the intake direction. The optical sensor is able to see how far away the disc is from the robot. So the robot drives until the optical sensor reads that the bot is right in front of a disc. This helps mitigate errors within the system by ensuring that the robot is in range to intake discs.

```
79  #function to position the robot in front of a disc
80  # uses the optical sensor and while loop conditionals
81  def movetodisc():
82      drivetrain.drive(FORWARD)
83      #robot moves forward until it has reached a disc
84      while not bottom_distance.object_distance(MM) < 20:
85          #checks if the disc is within 20 millimeters of the robot
86          wait(5, MSEC)
87      drivetrain.stop()
88      #if the while loop condition is fufilled, the drivetrain stops driving.
```

Intake Disc Function

Sensors Used: Distance Sensor, Line tracker

Programming structures:

If/else statements, while loops, for loops

Purpose of Function:

This function Intakes up to three discs with sensors such as the line tracker sensor helping ensure that no discs are unintentionally shot out. First the function uses the distance sensor to check if the discs are in range for intaking, if they are, the robot will begin to intake the discs. When the top line tracker has detected that the discs have reached the top of the intake, the motor stops to prevent the discs from being fired out.

```
93  #variables used in the intake disc function
94  stack = True
95  #the variable is assigned by the user, it tells the function whether or not the discs it is intaking is in a stack or not
96  #True denotes, yes, it is in a stack, while False means no stack
97  discstointake = 0
98  #This variable is also assigned by the user. It tells the function the amount of discs it needs to intake
99
100 #function to intake discs
101 #uses line and distence sensors, also uses while and for loop conditionals
102 def intakedisc(discstointake, stack):
103     if stack == True:
104     #checks if the discs are in a stack
105         while bottom_distance.object_distance(MM) < 40:
106         #checks if the discs are within 40 millimeters of the intake
107             intake_motor_group.spin(REVERSE)
108             #turns the roller in the Intake direction
109             wait(5, MSEC)
110         while not top_line_tracker.reflectivity(PERCENT)>80:
111         #checks if the disc is at the top of the intake
112             wait(5,MSEC)
113         intake_motor_group.stop()
114         #if the disc is at the top of the intake, the process is complete and
115     else:
116     #this part runs if the disc is not a stack
117         for x in range(discstointake):
118         #the loop repeats for every disc that needs to be intaked
119             movetodisc()
120             #the robot moves to the disc
121             while bottom_distance.object_distance(MM) < 40:
122             #if the robot is less than 40 millimeters away from the disc, it is able to intake it
123                 intake_motor_group.spin(REVERSE)
124                 #the robot intakes that disc and the for loop repeats for every disc needed
125                 wait(5, MSEC)
```

Drive to Coordinate Function

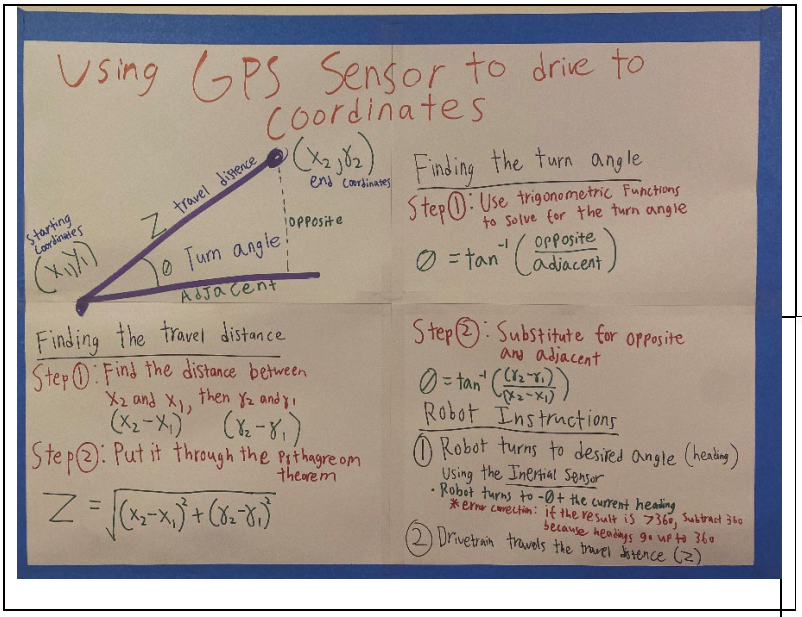
Purpose of Function:

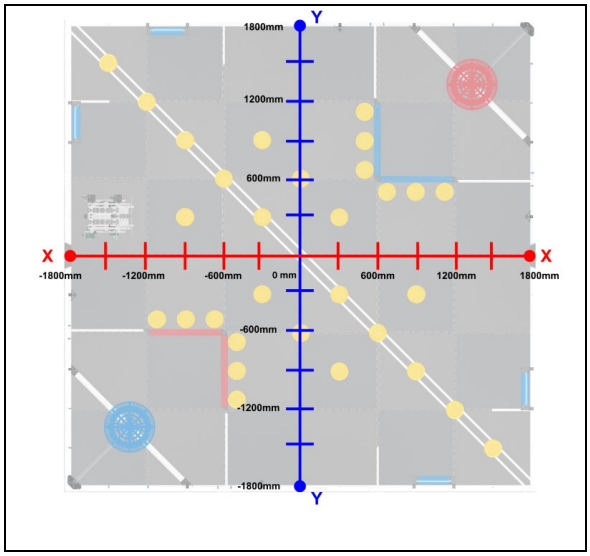
This function allows for the robot to drive to any position on the field, given the coordinates of the position. So because of this, I also use it to correct the position of my robot after driving over low goal

Sensors Used: GPS, Inertial sensor,

Programming structures:

barriers which usually result in inconsistencies within the robot's position. This function corrects this error by driving to a specific point, allowing the remainder of the autonomous routine to remain consistent.






```
#declaring the variables to be used in the drive to coordinate function
endposx = 0
endposy = 0
#end posy and x denote the coordinate that the function tells the robot to move to
#variables assigned by user

beginposx = 0
beginposy = 0
beginheading = 0
#These starting conditions collect info from the GPS sensor
#has the beginning coordinate position and orientation

turnangle = 0
turnheading = 0
travelDistance = 0
#These values are all CALCULATED by the function using trigonometric equations

Otherturn = False
#this boolean is assigned by the user to change how the robot can move in this function

#allows the robot to drive to anywhere on the field, given the desired end coordinate
def drivetocord(endposx,endposy, Otherturn):
#drive to coordinate function collects the desired endpoint and whether or not otherturn will be used

    beginposx = gps.x_position(MM)
    beginposy = gps.y_position(MM)
    beginheading = drivetrain.heading(DEGREES)
    #gets the values of the the robot's starting conditions when the function is called
    #this includes the GPS x and y positions of the robot and the robot heading as read in the inertial sensor

    travelDistance = math.sqrt((endposx-(beginposx))**2 + (endposy-(beginposy))**2)
    #calculates for travel distance using the pythagorean theorem

    turnangle = (math.degrees(math.atan2((endposy-beginposy), (endposx - beginposx))))
    #calculates the tangent using trigonometric functions
    #the tangent is the angle that the robot needs to turn to face the end position coordinate

    if Otherturn == False:
        turnheading = beginheading - turnangle
        #if the  otherturn boolean is not needed, then the turn headding is calculated normally
    else:
        turnheading = 360 -( beginheading - turnangle +35 )
        #if the otherturn boolean is needed, then it will make adjustments to the robot's turn so it remains accurate

    #To find out if otherturn is needed, it should be tested in the program

    #Headings go from 0-360
    #these corrections make if the turnheading goes over or under the range
    if turnheading > 360:
        turnheading = turnheading -360
    if turnheading < 0:
        turnheading = turnheading + 360

    #The robot turns to the calculated heading using the inertial sensor
    drivetrain.turn_to_heading(turnheading,DEGREES)
    #the robot travels the calculated distence
    drivetrain.drive_for(FORWARD,travelDistance,MM)
```