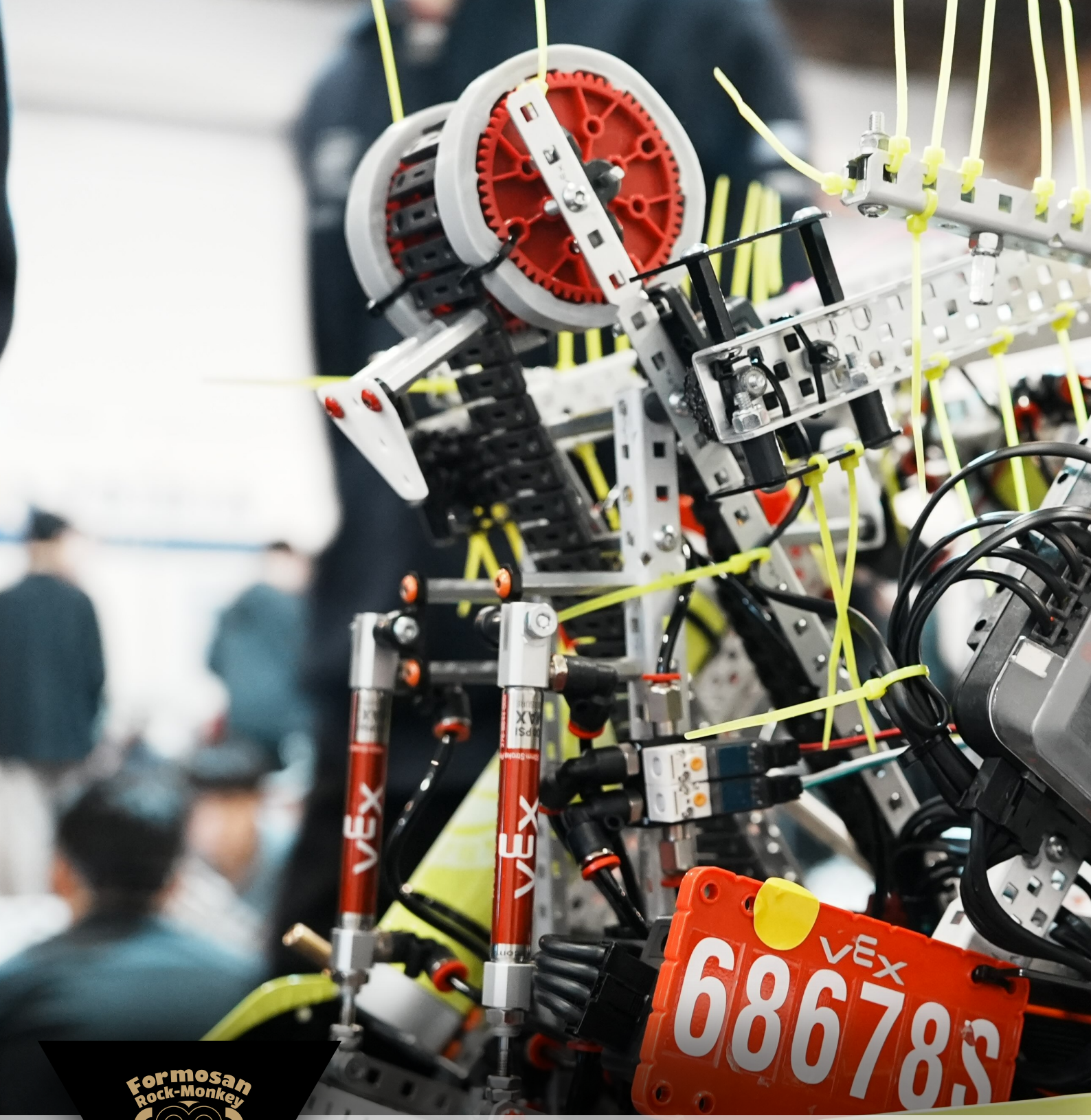



# VRC High Stake
# Virtual Skills Challenge
# Code Explanation

TEAM 68678S Formosan Rock-Monkey.

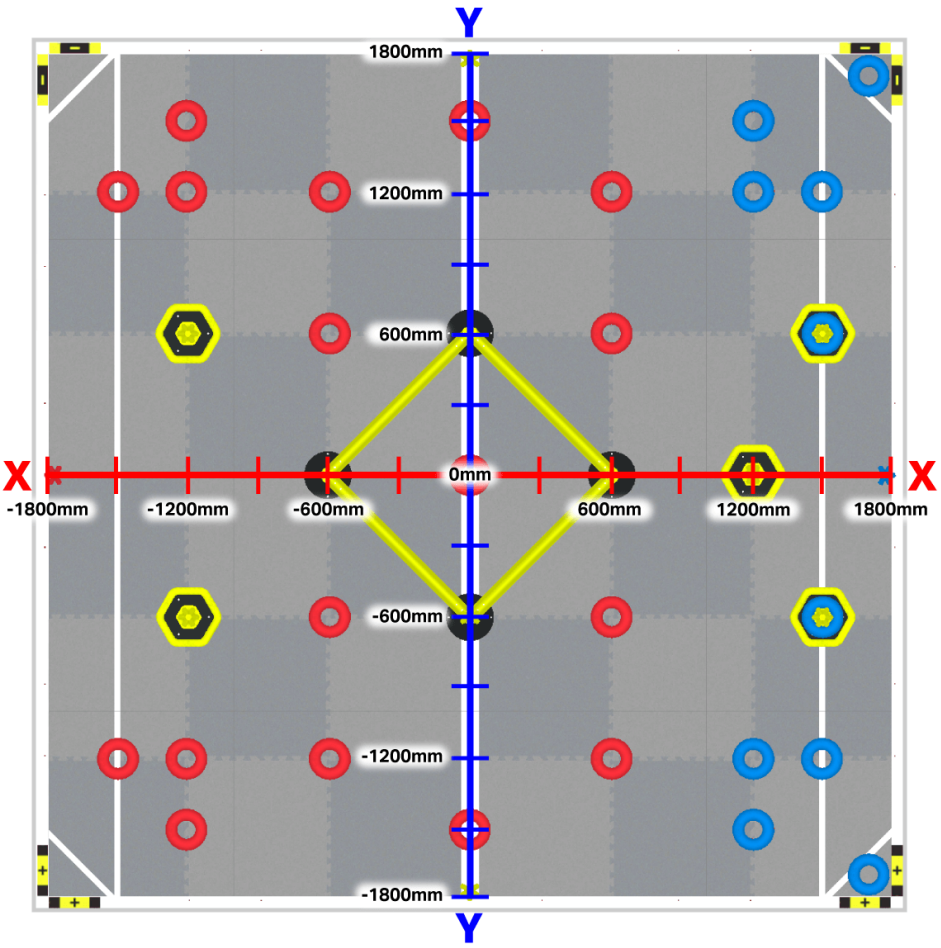
Fancy Design Limited | TAIPEI TAIWAN

LEE CHIA CHENG | JONATHAN CHENG | BRUCE LIN | SHIH CHEN HAN | SUN YIN TAO | KE YUN ZE

# CONTENTS

# Introduction to the GPS System

We integrated the reference coordinates from the VEX official website and applied functions to operate the GPS system. The GPS system is a core tool for robot navigation, responsible for providing precise real-time location data (x, y coordinates) and orientation (angle). By combining distance calculation and directional adjustments, the robot can efficiently perform various tasks, including moving to target positions, obstacle avoidance, and precisely controlling robotic arms for pick-and-place operations.




The V5RC High Stakes Field in VEXcode VR ranges from approximately -1800mm to 1800mm for the X and Y positions.

**Data Link : GPS Coordinates**

# Key Features of the System:

- **Position Tracking:**
  Real-time acquisition of the robot's precise spatial coordinates.

- **Angle Adjustment:**
  Formats the angle to ensure the robot faces the optimal direction.

- **Accurate Movement:**
  Utilizes distance and angle calculations to enable the robot to travel to its target via the shortest path.

- **Versatile Applications:**
  Supports both forward and reverse movements to meet the demands of different scenarios.

By integrating the GPS system with other sensors, the robot gains exceptional autonomous navigation capabilities, excelling in complex tasks such as object picking, scoring, and precise localization in the environment.

# Advantages of the GPS System:

1. **High Accuracy:**
   Provides precise coordinate and angle data, ideal for precision-based tasks.

2. **Easy Integration:**
   Can be combined with other sensors (e.g., optical or distance sensors) to enhance overall performance.

3. **Flexibility:**
   Supports both forward and reverse movements, adapting to different navigation needs.

# Practical Applications:

1. **Target Tracking:**

   Robots can use the GPS to determine their position and track a target object, such as facing to the location of rings or mobile goal

   1. face_coord(self, x, y, aiming, offset)
   2. face_angle(self, angle)

2. **Environmental Navigation:**

   In more complex environments, the robot can autonomously navigate using GPS based on maps and target locations.

   1. move_to_point(self, x, y)
   2. move_to_point_backward(self, x, y)

3. **Task Execution:**

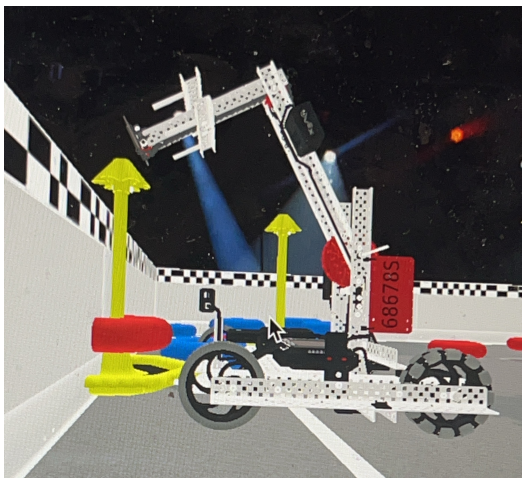
   Working in conjunction with other components (e.g., arms, grippers), the robot can complete tasks such as picking and placing objects or scoring.

# Game strategy

The primary objective of the 60-second vex VR skill simulation is to strategically collect and place game elements (rings, mobile goals) to achieve the highest score within the allotted time. The robot must operate using python or blocks programmed instructions to efficiently complete tasks following the scoring system below:

| | |
|---|---|
| Each Ring Scored on a Stake | 1 Point* |
| Each Top Ring on a Stake | 3 Points* |
| Climb - Level 1 | 3 Points |
| Climb - Level 2 | 6 Points |
| Climb - Level 3 | 12 Points |
| Mobile Goal Placed in a Corner | 5 Points |

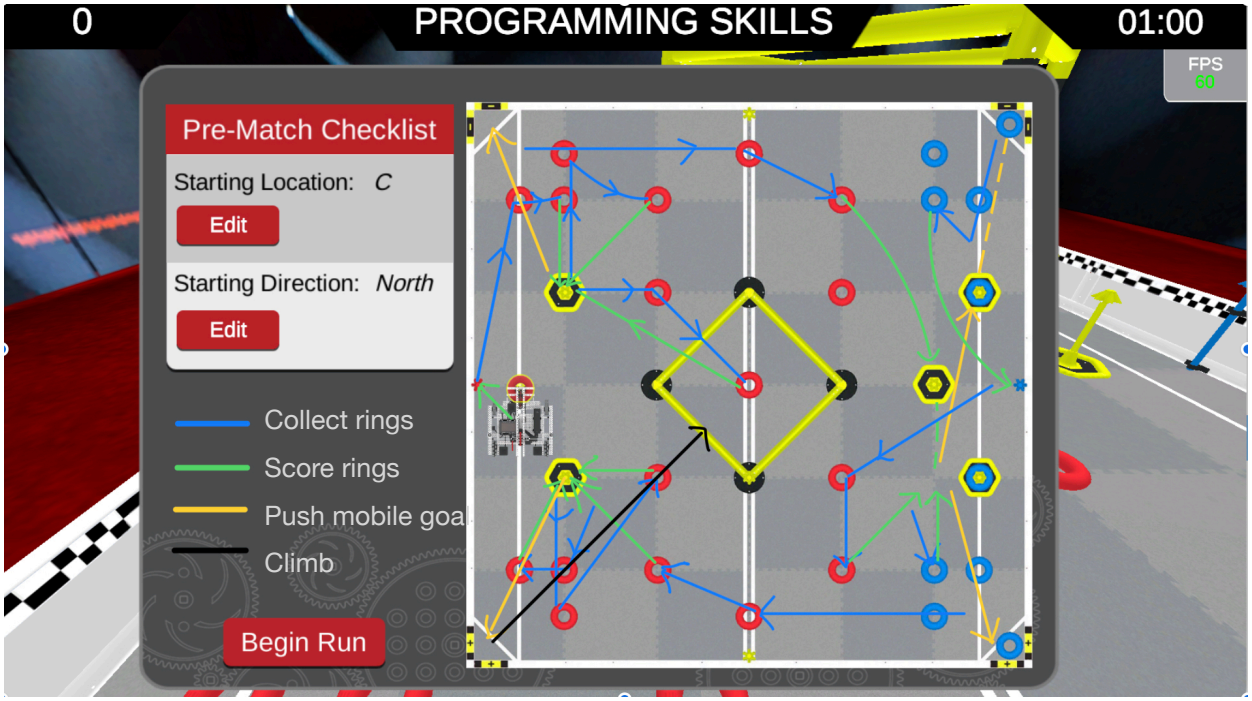
Within the limited time, our primary goal is to achieve the highest possible score in the VR skill simulation. Therefore, our strategy focuses on accurately placing the top ring on each stake and ensuring that the mobile goal is positioned in each corner. This approach maximizes the ring score, which is crucial for achieving high scores.




Initially, we attempted to remove the blue ring to gain extra top rings.
However, As seen in the picture, although we have removed the blue ring and placed another red ring inside the mobile goal, the "soul" of the blue ring still remains, preventing the red ring from being counted for points.we quickly realized that this strategy was not feasible and wasted valuable time, so we decided to abandon it.

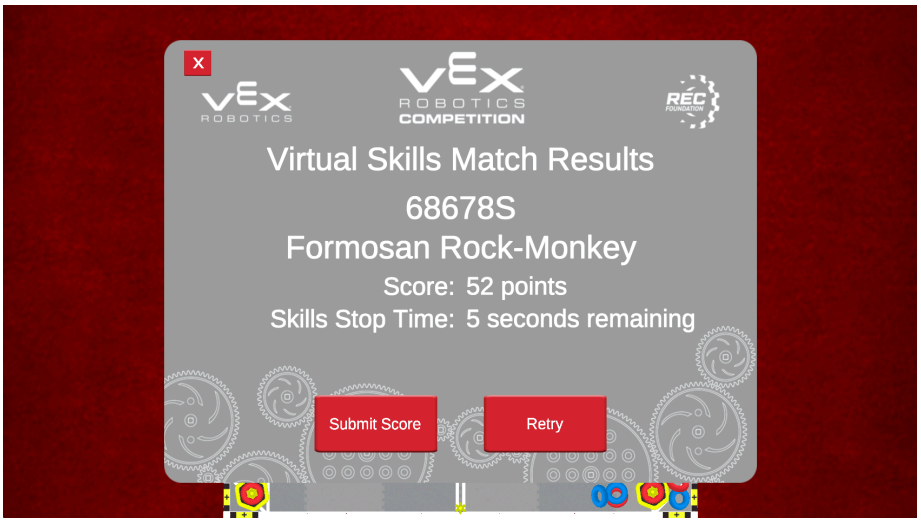
Consequently, we shifted our focus to achieving a Level 1 climb. We are now concentrating on fine-tuning the robot's movement path and program parameters to ensure that we can reliably reach the climbing platform within the 60 seconds.

# Robot pathing




We start our robot at position C, facing north. We place the red alliance stake in the starting area and complete a full mobile goal with 6 rings, including the one in the middle, then position it in the top-left corner.

Next, we move to the right side of the field and push the empty mobile goal downward creating a shorter path for later scoring. We then push the mobile goal with the blue ring to the top-right corner and collect the ring from the corner, placing it on the blue alliance stake. Afterward, we add a total of 5 rings to the mobile goal we just moved and place it in the bottom-right corner to achieve control of the third corner.



Finally, we fill the last empty mobile goal with 6 rings and place it in the last corner at the bottom left. Then, we go under the ladder to perform our last action: a Level 1 climb.

By following this path, we scored 52 points with 5 seconds remaining and secured 14th place in the VR skill competition. Unfortunately, after a few weeks, we dropped to 25th place. Nevertheless, we successfully achieved our objectives: placing a mobile goal in all four corners, scoring as many top rings as possible, and completing a Level 1 climb.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 | 53 | 2 | 38535D | R2-DE2 | Glenbrook South High School | Illinois | United States |
| 17 | 53 | 1 | 355A | 2Xstream Robotics | Fox Valley Robotics | Illinois | United States |
| 18 | 53 | 0 | 41103B | ASDF | Pedare Christian College | Australia | Australia |
| 19 | 53 | 0 | 60172W | WHAM | Lake Park High School | Illinois | United States |
| 20 | 53 | 0 | 10821D | Delta | Fundacion IQ Tecnologia | Colombia | Colombia |
| 21 | 52 | 26 | 35211A | Saints A | CREAN LUTHERAN HIGH SCHOOL | California - Region 4 | United States |
| 22 | 52 | 13 | 2813B | NOVA | CHINGSHIN ACADEMY | Chinese Taipei | Taiwan |
| 23 | 52 | 9 | 85301V | Generic | The International School of Macao | Macau | Macau |
| 24 | 52 | 5 | 67732B | PSGD Aeolus | Pobalscoil Ghaoth Dobhair | Ireland | Ireland |
| 25 | 52 | 5 | 68678S | Formosan Rock-Monkey | FANCY DESIGN LIMITED | Chinese Taipei | Taiwan |
| 26 | 52 | 4 | 6408H | Hollow | Heritage Woods Secondary | British Columbia (BC) | Canada |
| 27 | 52 | 1 | 3204P | SPC Bots Paul | St Peter's College | New Zealand | New Zealand |

# The main function explanation

## Angle Normalization Function for Range [-180, 180]

The purpose of this code is to ensure that any angle value is confined within the range of -180 to 180 degrees. This guarantees that angle values remain within a reasonable range, preventing excessively large or small numbers.

```
def format_angle(a):
 sign = 1
 if a < 0:
   sign = -1
 else:
   sign = 1
 positive_a = abs(a)
 mod = positive_a % 360


 if mod <= 180:
  return sign * mod
 else:
  return sign * (mod - 360)
```

---

## Coordinate Class for Position and Orientation Representation

It is used to represent coordinates in a two-dimensional space along with their direction (angle).

```
class Coord:
  def __init__(self, x, y, theta=0):
     self.x = x
     self.y = y
     self.theta = theta
```

---

## Singleton Pattern Implementation for Chassis Class

This code implements a Singleton Pattern for a chassis class, ensuring that throughout the program's runtime, there can only be one instance of the chassis class.
This means that no matter how many times objects of the chassis class are created, they will all point to the same instance. (An instance is a specific object created from a class that can hold data and functions, and it can be manipulated within the program.)

```
class Chassis:
  __instance__ = None


  def __init__(self):
    if Chassis.__instance__ is None:
      Chassis.__instance__ = self  #To create a unique instance, you can use the Singleton design pattern.
    else:
      raise Exception("You cannot create another Chassis class")  # Throw an exception
```

---

## face_angle(self, angel)

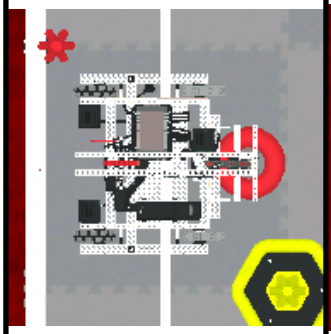
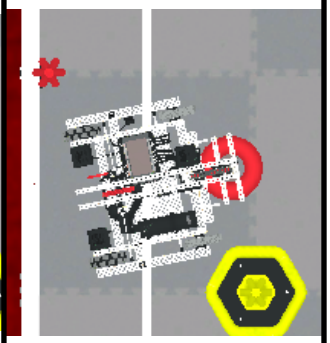
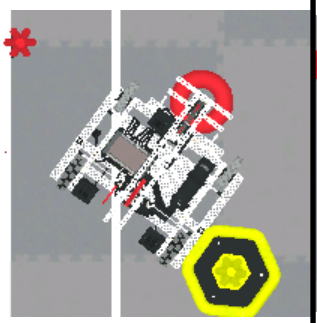
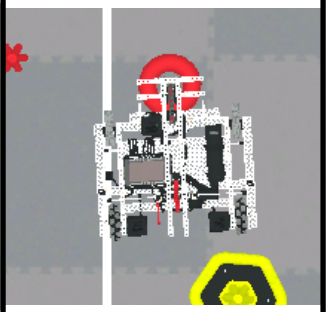
First, the method reads the robot's current position and orientation (gps.x_position, gps.y_position, gps.heading).Then, it calculates the difference between the target angle (angle) and the current angle (position.theta), and normalizes it to be between -180° and 180°.Finally, the robot rotates based on the calculated angle using drivetrain.turn_for（）, aligning the robot towards the target angle.

```
  def face_angle(self, angle):
    drivetrain.set_turn_velocity(100, PERCENT)
    position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
    target_angle = format_angle(angle - position.theta)
    drivetrain.turn_for(RIGHT, target_angle, DEGREES)
```

---

## face_coord(self, x, y, aiming, offset)

#The purpose of this code is to adjust the robot's orientation based on the given target coordinates (x, y) and the aiming parameter, allowing the robot to choose whether to face the target coordinates with its front or rear. This helps the robot make the necessary positioning and adjustments according to specific task requirements.

| Starting position | Face_coord (0,0) | Face_coord(0,1600) | Face_coord(1600,1600) |
|---|---|---|---|
|  |  |  |  |

**Video link:face_coord(self, x, y, aiming, offset)**

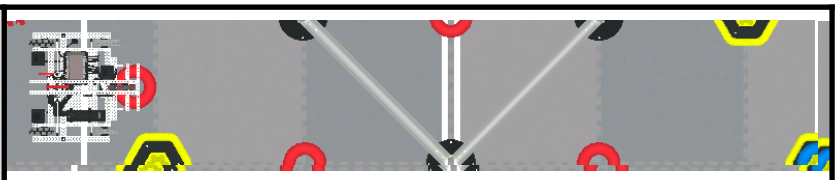
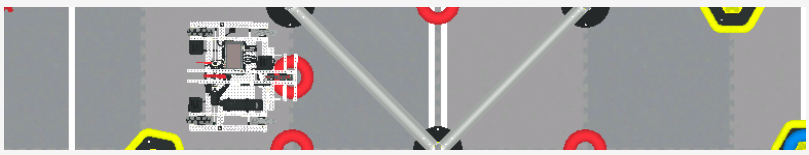
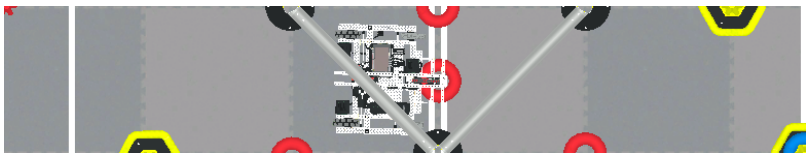
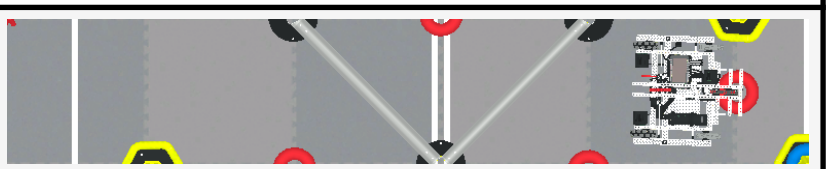
```
def face_coord(self, x, y, aiming, offset):
    drivetrain.set_turn_velocity(100, PERCENT)
    position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
    dx = x - position.x
    dy = y - position.y
    target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
    if aiming:
     # face coord backwards
    drivetrain.turn_for(RIGHT, float(format_angle(180 + target_angle - position.theta + offset)), DEGREES)
    else:
     # face the coord
     drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta + offset)), DEGREES)
```

## move_to_point(self, x, y)

By using the distance formula, the robot calculates the straight-line distance from its current position to the target position. This ensures that the robot takes the shortest path when moving to the target location.

**Video link: move_to_point(self, x, y)**

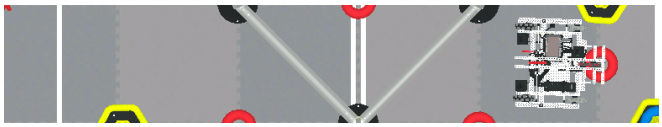
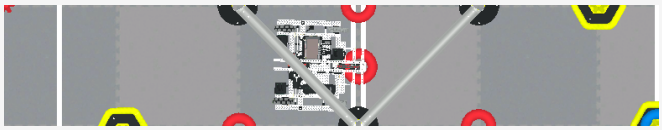
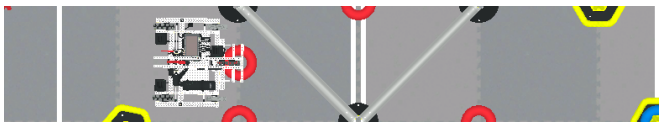
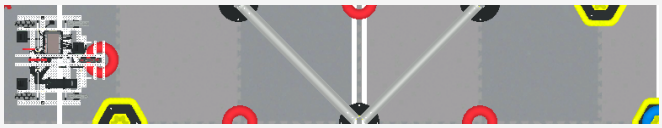
```
def move_to_point(self, x, y):#Move forward
    drivetrain.set_drive_velocity(100, PERCENT)
    position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
    dx = x - position.x
    dy = y - position.y
    dist = (math.sqrt(dx**2 + dy**2))-100
    target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
    drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta)), DEGREES)
    drivetrain.drive_for(FORWARD, dist, MM)
```

| | |
|---|---|
| **Starting position** |  |
| **move_to_point(-600,-300)** |  |
| **move_to_point(0,-300)** |  |
| **move_to_point(1200,-300)** |  |

## move_to_point_backward(self, x, y)

Using the distance formula, the robot calculates the straight-line distance from its current position to the target position. This ensures that the robot follows the shortest path when moving to the target location.

```
def move_to_point_backward(self, x, y):#Reverse
    drivetrain.set_drive_velocity(100, PERCENT)
    position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
    dx = x - position.x
    dy = y - position.y
    dist = (math.sqrt(dx**2 + dy**2))-100
```

| | |
|---|---|
| **Starting position** |  |
| **move_to_point_backward(0,-300)** |  |
| **move_to_point_backward(-600,-300)** |  |
| **move_to_point_backward(-1300,-300)** |  |

## Robot Movement and Singleton Pattern for Scoring Mechanism

The purpose of this code is to move the robot to a specified (x, y) coordinate. It achieves this through the following steps:

1. Calculate the distance and direction between the target coordinate and the current position.

2. Rotate the robot according to the target angle so that it faces the target.

3. Move the robot to the target position.

```
    target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
    drivetrain.turn_for(RIGHT, float(format_angle(180 + target_angle - position.theta)), DEGREES)
    drivetrain.drive_for(REVERSE, dist, MM)
class Score:
  __instance__ = None


  def __init__(self):
    if Score.__instance__ is None:
      Score.__instance__ = self
      self.chassis = Chassis()
      self.Score = Score()
    else:
      raise Exception("You cannot create another Score class")
```

## Singleton Pattern Implementation for Intake and Robot Classes

This code defines static methods for controlling a robot's movement and orientation. It uses the Singleton pattern to ensure only one instance of the Robot class exists. The methods include controlling the robot’s rotation to a specific angle or target coordinates, moving to a target point, and performing tasks like taking a ring. These methods provide easy access to the robot's actions without needing to create multiple instances.

```
class Intake:
  __instance__ = None

  def __init__(self):
    if Intake.__instance__ is None:
      Intake.__instance__ = self
    else:
      raise Exception("You cannot create another Intake class")

class Robot:
  __instance__ = None

  def __init__(self):
    if Robot.__instance__ is None:
      Robot.__instance__ = self
      self.chassis = Chassis()
      self.intake = Intake()
    else:
      raise Exception("You cannot create another Robot class”)
```

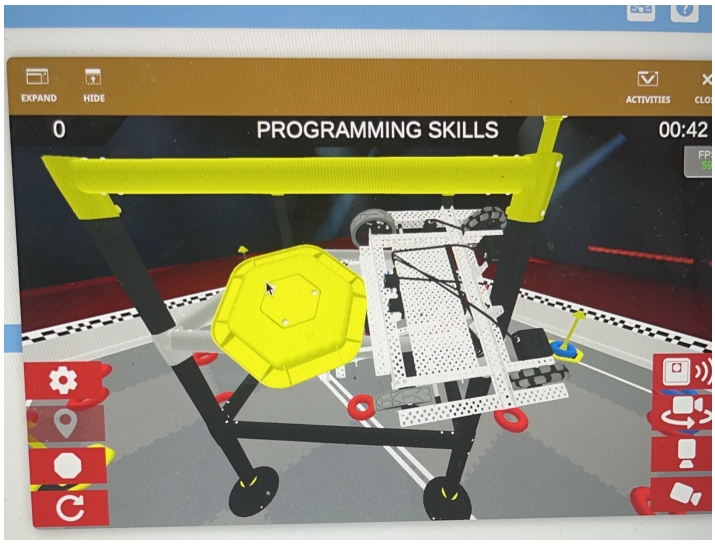
## Static Methods for Robot Control and Singleton Instance Access

This code defines static methods for the Robot class to control its movement and orientation. The get_instance() method ensures a single instance of the robot. The face_angle(angle) and face_coord(x, y, aiming, offset) methods rotate the robot to a specific angle or target coordinates. The move_to_point(x, y) and move_to_point_backward(x, y) methods move the robot forward or backward to target coordinates. The move_to_point_take_ring(x, y) method moves the robot and interacts with the Score component to take a ring.

```
@staticmethod
def get_instance():
    if Robot.__instance__ is None:
        Robot()
    return Robot.__instance__


@staticmethod
def face_angle(angle):#Facing angle
    Robot.__instance__.chassis.face_angle(angle)


@staticmethod
def face_coord(x, y, aiming=False, offset=0):#Facing the target point
    Robot.__instance__.chassis.face_coord(x, y, aiming, offset)


@staticmethod
def move_to_point(x, y):#Move to the target point
    Robot.__instance__.chassis.move_to_point(x, y)


@staticmethod
def move_to_point_backward(x, y):#Reverse to the target point
    Robot.__instance__.chassis.move_to_point_backward(x, y)


@staticmethod
def move_to_point_take_ring(x,y):
    Robot._instance_.Score.move_to_point_take_ring(x,y)
```

# Challenges and Learning

In using Python programming for the GPS system in an online robotics simulation, several challenges were encountered. One of the biggest issues was coordination errors, causing the robot to potentially fall out of the field or behave unexpectedly. There were also bugs in the simulation, such as the robot failing to execute commands correctly or acting illogically.




This process provided valuable learning opportunities, particularly in debugging and optimizing the simulation. By adjusting the accuracy of coordinate calculations, improving error handling, and refining the robot's movement logic, these challenges were successfully overcome. Ultimately, it taught me the importance of attention to detail in robotics programming, with each error serving as a learning opportunity.

**YouTube link : [VEXcode VR Virtual Skills Challenge - Team 68678S]**

# Final Code

## Code File link : final code

## Final show link : side view | top view

```
#The original configuration of the Region VEXcode Generated Robot.
import math
import random
from vexcode_vrc import *
from vexcode_vrc.events import get_Task_func

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

arm_motor = Motor("ArmMotor", 3)
pusher_motor = Motor("PusherMotor", 8)
front_distance = Distance("FrontDistance", 6)
distance = front_distance
front_optical = Optical("FrontOptical", 5)
gps = GPS("GPS", 17)
drivetrain = Drivetrain("drivetrain", 0)
pusher_rotation = Rotation("PusherRotation", 19)
ai_vision = AiVision("aiVision", 7)

# 'Enum' class for AI Vision GameElements
class GameElements:
  MOBILE_GOAL = 0
  RED_RING = 1
  BLUE_RING = 2

#endregion VEXcode Generated Robot Configuration
#region VEXcode Generated Robot Configuration
import math
import random
from vexcode_vrc import *
from vexcode_vrc.events import get_Task_func


#region helper functions


def format_angle(a):
  sign = 1
  if a < 0:
    sign = -1
  else:
    sign = 1
  positive_a = abs(a)
  mod = positive_a % 360

  if mod <= 180:
   return sign * mod
```

```
    else:
      return sign * (mod - 360)

class Coord:
    def __init__(self, x, y, theta=0):
        self.x = x
        self.y = y
        self.theta = theta

class Chassis:
    __instance__ = None

    def __init__(self):
        if Chassis.__instance__ is None:
            Chassis.__instance__ = self
        else:
            raise Exception("You cannot create another Chassis class")  # Throw an exception


    def face_angle(self, angle):
        drivetrain.set_turn_velocity(100, PERCENT)
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
        target_angle = format_angle(angle - position.theta)
        drivetrain.turn_for(RIGHT, target_angle, DEGREES)

def face_coord(self, x, y, aiming, offset):
        drivetrain.set_turn_velocity(100, PERCENT)
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
        dx = x - position.x
        dy = y - position.y
        target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
        if aiming:

            drivetrain.turn_for(RIGHT, float(format_angle(180 + target_angle - position.theta + offset)), DEGREES)
        else:

            drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta + offset)), DEGREES)


    def move_to_point(self, x, y):
        drivetrain.set_drive_velocity(100, PERCENT)
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
        dx = x - position.x
        dy = y - position.y
        dist = (math.sqrt(dx**2 + dy**2))-100
        target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
        drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta)), DEGREES)
        drivetrain.drive_for(FORWARD, dist, MM)


    def move_to_point_backward(self, x, y):
        drivetrain.set_drive_velocity(100, PERCENT)
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
        dx = x - position.x
```

```
        dy = y - position.y
        dist = (math.sqrt(dx**2 + dy**2))-100
        target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
        drivetrain.turn_for(RIGHT, float(format_angle(180 + target_angle - position.theta)), DEGREES)
        drivetrain.drive_for(REVERSE, dist, MM)

class Score:
    __instance__ = None

    def __init__(self):
        if Score.__instance__ is None:
            Score.__instance__ = self
            self.chassis = Chassis()
            self.Score = Score()
        else:
            raise Exception("You cannot create another Score class")

    def move_to_point_take_ring(self,x,y):
        arm_motor.set_velocity(100, PERCENT)
        drivetrain.set_drive_velocity(100, PERCENT)
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
        dx = x - position.x
        dy = y - position.y
        dist = (math.sqrt(dx**2 + dy**2))-100

        target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
        arm_motor.spin_for(FORWARD,100,DEGREES)
        drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta)), DEGREES)
        drivetrain.drive_for(FORWARD, dist, MM)
        arm_motor.spin_for(FORWARD,-100,DEGREES)

    def move_to_point_score_ring(self,x,y):
        arm_motor.set_velocity(100, PERCENT)
        drivetrain.set_drive_velocity(100, PERCENT)
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
        dx = x - position.x
        dy = y - position.y
        dist = (math.sqrt(dx**2 + dy**2))-100

        target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
        arm_motor.spin_for(FORWARD,400,DEGREES)
        drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta)), DEGREES)
        drivetrain.drive_for(FORWARD, dist, MM)
        pusher_motor.spin_for(FORWARD,790,DEGREES,wait=True)
        pusher_motor.spin_for(FORWARD,-790,DEGREES,wait=True)


class Intake:
    __instance__ = None

    def __init__(self):
        if Intake.__instance__ is None:
            Intake.__instance__ = self
        else:
```

```
            raise Exception("You cannot create another Intake class")



class Robot:
    __instance__ = None

    def __init__(self):
        if Robot.__instance__ is None:
            Robot.__instance__ = self
            self.chassis = Chassis()
            self.intake = Intake()
        else:
            raise Exception("You cannot create another Robot class")

    @staticmethod
    def get_instance():
        if Robot.__instance__ is None:
            Robot()
        return Robot.__instance__

    @staticmethod
    def face_angle(angle):
        Robot.__instance__.chassis.face_angle(angle)

    @staticmethod
    def face_coord(x, y, aiming=False, offset=0):
        Robot.__instance__.chassis.face_coord(x, y, aiming, offset)

    @staticmethod
    def move_to_point(x, y):
        Robot.__instance__.chassis.move_to_point(x, y)

    @staticmethod
    def move_to_point_backward(x, y):
        Robot.__instance__.chassis.move_to_point_backward(x, y)

    @staticmethod
    def move_to_point_take_ring(x,y):
        Robot._instance_.Score.move_to_point_take_ring(x,y)




def main():
# robot setting
 robot = Robot.get_instance()
 drivetrain.set_drive_velocity(100, PERCENT)
 drivetrain.set_turn_velocity(100, PERCENT)
 pusher_motor.set_velocity(100, PERCENT)
 arm_motor.set_velocity(100, PERCENT)
```

```
 arm_motor.spin_for(FORWARD,400,DEGREES,wait=False)
 pusher_motor.spin_for(FORWARD,400,DEGREES,wait=False)

# red alliance stake
 robot.face_coord(-1800,0, aiming=False, offset=0)
 pusher_motor.spin_for(FORWARD,650,DEGREES)
 pusher_motor.spin_for(REVERSE,650,DEGREES,wait=False)
 arm_motor.spin_for(REVERSE, 400,DEGREES,wait=False)


# take rings
 robot.face_coord(-1500,1200, aiming=False, offset=0)
 robot.move_to_point(-1550,1200)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 100,DEGREES,wait=False)

 robot.move_to_point(-1200,1300)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 350,DEGREES,wait=False)

# score rings
 robot.move_to_point(-1050,650)
 pusher_motor.spin_for(FORWARD,650,DEGREES)
 pusher_motor.spin_for(REVERSE,650,DEGREES,wait=False)
 drivetrain.drive_for(REVERSE,400,MM)

# take rings
 arm_motor.spin_for(REVERSE, 350,DEGREES,wait=False)
 robot.face_coord(-1000,1500, aiming=False, offset=0)
 robot.move_to_point(-1150,1500)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 100,DEGREES,wait=False)

 robot.move_to_point(-580,1280)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 350,DEGREES,wait=False)

# score rings
 robot.move_to_point(-1050,500)
 pusher_motor.spin_for(FORWARD,650,DEGREES)
 pusher_motor.spin_for(REVERSE,650,DEGREES,wait=False)
 drivetrain.drive_for(REVERSE,300,MM)

# take rings
 arm_motor.spin_for(REVERSE, 350,DEGREES,wait=False)
 robot.face_coord(-1000,1500, aiming=True , offset=0)
 robot.move_to_point(-600,600)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 100,DEGREES,wait=False)

 robot.move_to_point(0,0)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 100,DEGREES,wait=False)
```

```
# 6rings+corner 1
 robot.move_to_point(-900,300)
 arm_motor.spin_for(FORWARD, 300,DEGREES,wait=False)
 robot.move_to_point(-1200,600)
 pusher_motor.spin_for(FORWARD,650,DEGREES)
 pusher_motor.spin_for(REVERSE,650,DEGREES,wait=False)
 robot.move_to_point(-1700,1500)
 drivetrain.drive_for(REVERSE,300,MM)


# take rings
 arm_motor.spin_for(REVERSE, 300,DEGREES,wait=False)
 robot.face_coord(0,1600, aiming=False , offset=0)
 robot.move_to_point(0,1600)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 100,DEGREES,wait=False)

 robot.move_to_point(600,1300)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 400,DEGREES,wait=False)



# score rings+clean path
 robot.move_to_point(1300,600)
robot.move_to_point(1300,-700)
 pusher_motor.spin_for(FORWARD,650,DEGREES)
 pusher_motor.spin_for(REVERSE,650,DEGREES,wait=False)
 drivetrain.drive_for(REVERSE,300,MM)

# corner
 drivetrain.drive_for(REVERSE,300,MM)
 robot.face_coord(1500,600, aiming=False , offset=0)
 robot.move_to_point(1500,600)
 robot.move_to_point(1600,1600)
 drivetrain.drive_for(REVERSE,600,MM)

# take rings
 arm_motor.spin_for(REVERSE, 400,DEGREES,wait=False)
 robot.move_to_point(1200,1200)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 400,DEGREES,wait=False)


#remove blue ring (blue alliance)
 robot.face_coord(1700,0, aiming=False , offset=0)
 robot.move_to_point(1650,100)
 pusher_motor.spin_for(FORWARD,300,DEGREES)
 pusher_motor.spin_for(FORWARD,400,DEGREES,wait=False)
 drivetrain.drive_for(REVERSE,150,MM)

# take rings
```

```
 robot.face_coord(1200,0, aiming=False , offset=0)
  pusher_motor.spin_for(REVERSE,650,DEGREES,wait=False)
 robot.move_to_point(1200,0)
 arm_motor.spin_for(REVERSE, 400,DEGREES,wait=False)
 robot.move_to_point(600,-600)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 100,DEGREES,wait=False)

 robot.move_to_point(600,-1200)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 400,DEGREES,wait=False)
 robot.move_to_point(1100,-600)

# score rings
 pusher_motor.spin_for(FORWARD,650,DEGREES)
 pusher_motor.spin_for(REVERSE,650,DEGREES,wait=False)

 # take rings
 drivetrain.drive_for(REVERSE,50,MM)
 robot.face_coord(1200,-1200, aiming=False , offset=0)
 arm_motor.spin_for(REVERSE, 400,DEGREES,wait=False)
 robot.move_to_point(1200,-1200)

 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 400,DEGREES,wait=False)

# remove blue ring (mobile goal)
 robot.move_to_point(1100,-600)
 robot.move_to_point(1700,-1600)
 pusher_motor.spin_for(FORWARD,250,DEGREES)
 drivetrain.drive_for(REVERSE,100,MM)
 pusher_motor.spin_for(FORWARD,400,DEGREES,wait=False)

 # take rings
  arm_motor.spin_for(REVERSE, 400,DEGREES,wait=False)
 robot.move_to_point(0,-1550)
 pusher_motor.spin_for(REVERSE,650,DEGREES,wait=False)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 100,DEGREES,wait=False)

 robot.move_to_point(-600,-1200)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 400,DEGREES,wait=False)


# score rings
 robot.move_to_point(-1200,-600)
 pusher_motor.spin_for(FORWARD,650,DEGREES)
 pusher_motor.spin_for(REVERSE,650,DEGREES,wait=False)


# take rings
```

```
 drivetrain.drive_for(REVERSE,300,MM)
 arm_motor.spin_for(REVERSE, 400,DEGREES,wait=False)
 robot.move_to_point(-1200,-1200)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 100,DEGREES,wait=False)

 drivetrain.drive_for(FORWARD,500,MM)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 400,DEGREES,wait=False)

# score rings
 robot.move_to_point(-1300,-600)
 pusher_motor.spin_for(FORWARD,650,DEGREES)
 pusher_motor.spin_for(REVERSE,650,DEGREES,wait=False)

# take rings
 drivetrain.drive_for(REVERSE,300,MM)
 arm_motor.spin_for(REVERSE, 400,DEGREES,wait=False)
 robot.move_to_point(-1000,-1550)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 100,DEGREES,wait=False)

 robot.move_to_point(-700,-600)
 arm_motor.spin_for(REVERSE, 100,DEGREES)
 arm_motor.spin_for(FORWARD, 400,DEGREES,wait=False)

# 6rings+corner 2
 robot.move_to_point(-1300,-600)
 pusher_motor.spin_for(FORWARD,650,DEGREES)
 pusher_motor.spin_for(REVERSE,650,DEGREES,wait=False)
 arm_motor.spin_for(FORWARD, 400,DEGREES,wait=False)

 robot.move_to_point(-1600,-1500)
 drivetrain.drive_for(REVERSE,300,MM)

# climb
 robot.face_coord(-450,-250, aiming=False , offset=0)
 drivetrain.drive_for(FORWARD,1650,MM)
 arm_motor.spin_for(REVERSE, 400,DEGREES)



 # VR threads — Do not delete
vr_thread(main)
```