High School VEXcode VR Skills Challenge

Created by: Christopher Tang and Victor Peel
Team: 11475D Liberty Patriots
Location: Bakersfield, California Region 3

First Impressions:

- The simulation is pretty well programmed
- The physics of the grabbing and shooting have an element of randomness
- The arm motor is really slow because of the gear ratio

Observations About The Physics Engine:

- Engine is framerate dependent, meaning the speed of the computer affects the performance of the simulation.
- The triballs match loads are stored below the field. They are visible when the robot accidentally phased through the field floor. (See last bullet).
- Intake can grab triballs by just being in close proximity.

- Adjusting the angle of the intake and combining it with both the forward velocity of the robot and the engine's gravitational acceleration can dramatically increase the triball's kinetic energy
- Some of the hitboxes are almost two dimensional (like the net) meaning you can clip through them through the side
- Triballs take time to settle when spawning – most likely because they are pushed out of the floor when the simulation is set. This will affect the run if it is started too quickly
- There is some random element to the triball movement/hitbox collisions. The reason is unknown. May be due to FPS, or inherent collision mechanics.
- If the robot has a bad interaction with the net or field wall, it may flip over and clip through/get stuck in the floor.

Challenges We Faced While Programming

Problem:	Solution:
Launch Power: The triballs' shoot/launch distance was very small, and scoring them under the net required lots of time-consuming robot movement.	**By analyzing online videos and experimenting with different shot positions**, we came to the conclusion that **the speed of the triball is affected by the robot's momentum**. By shooting while moving, adjusting the angle of the shot, and going over the middle pvc pipe, we could increase the speed and power of our shots.
Optimal Routing: We had very little time (1 minute) to travel across large areas of the field.	**By trial and error**, our testing of different routes allowed us to **increase the number of points** we were able to score.
Imprecise Turning: Part of our program involves going over the rounded pipe in the middle. After going over the middle, the orientation of the robot would be affected, and all subsequent turns the robot made would be offset and inaccurate	**By using the gyro sensor mounted on the robot**, we were able to program the robot to turn with respect to the absolute orientation of the field instead of the relative orientation of the robot through the `drivetrain.turn_to_heading()` function, thus **eliminating the inaccurate rotation caused by the bumps on the field.**

Massive program size: After our program had been developed for some time, it stretched over 300 lines. Whenever we wanted to change the travel distance of the robot for all the match loads, we'd have to go through the code and change every individual number.	**By wrapping the repeated match load code inside a function**, we could repeat the duplicated code with a loop, decreasing code size. We also standardized travel distance with one variable, which we could collectively change, **increasing our code readability.**

How this improved our coding skills:

Programming in this challenge has brought several important lessons to our attention. First, having to stress-test and revise our code over and over again is common. Second, you can only control the things you can control. Random will always be a factor. This has increased our familiarity with Python syntax that can help us program for actual robots. By constantly starting and observing the runs we did, we got a lot better at identifying where in the program the simulation is currently running. This helped a lot with debugging, as we didn't have to look long for where the problem occurred. The time spent has also taught us that there is always a way to improve and grow.

How this applies to the competition:

Having to test a program over and over again is not something that only holds true for the virtual simulation. In the competitions we attend in real life, the skills autonomous and alliance autonomous programs all have to be run multiple times with multiple changes and revisions to ensure accuracy and precision. The coding process taught us about the patience required to solve problems. We found that it was very beneficial to think from multiple perspectives and try new ideas. This lesson applies not only to the physical VEX competition, but also to the many other endeavors we may choose to pursue in the future.

The code:

Python

```
#region VEXcode Generated Robot Configuration
import math
```

```python
import random
from vexcode_vrc import *
from vexcode_vrc.events import get_Task_func


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


drivetrain = Drivetrain("drivetrain", 0)
arm_motor = Motor("ArmMotor", 3)
rotation = Rotation("Rotation", 7)
intake_motor = Motor("IntakeMotor", 8)
optical = Optical("Optical", 11)
gps = GPS("GPS", 20)


#endregion VEXcode Generated Robot Configuration
# ----------------------------------------
#
#   Project:      VexCode VR Project
#   Author:       Chirstopher Tang with help from Victor Peel
#   Created:      11/15/2023
#   Description:  VEXcode VR Python Project (up to 75 pts)
#
# ----------------------------------------


# function to configure the settings before the run - we maximize the power
from the motors
def setup():
  # Motor Configuration
  drivetrain.set_drive_velocity(100, PERCENT)
  drivetrain.set_turn_velocity(100, PERCENT)
  arm_motor.set_velocity(100, PERCENT)
  intake_motor.set_velocity(100, PERCENT)
  # Gyro Configuration
  drivetrain.set_rotation(0, DEGREES)
```

```
# scores the two red triballs - one preload which we unload into the goal and
one which we grab and throw into the goal
def preloads():
  # extend arm, drive to goal, and score preload into goal
  arm_motor.spin(FORWARD)
  wait(0.45, SECONDS)
  drivetrain.drive_for(FORWARD, 50, INCHES)
  intake_motor.spin(REVERSE)
  drivetrain.turn_to_heading(270, DEGREES)


  #grab red preload which is placed in spot 4, turn, and score into goal again
  drivetrain.turn_to_heading(0, DEGREES)
  intake_motor.spin(FORWARD)
  drivetrain.drive_for(FORWARD, 5, INCHES)
  intake_motor.spin(REVERSE)
  drivetrain.turn_to_heading(275, DEGREES)
  wait(0.48, SECONDS)


# scores the 6 triballs in the center.
def center_triballs():
  # pick up and score the closest triball to the robot green triball. This
traverses in between the center triball and the one above it.
  drivetrain.turn_to_heading(0, DEGREES)
  intake_motor.spin(FORWARD)
  drivetrain.turn_to_heading(60, DEGREES)
  drivetrain.drive_for(FORWARD, 25, INCHES)
  # begin the shot release now so it exits the intake with maximum force
  intake_motor.spin(REVERSE)
  drivetrain.turn_to_heading(90, DEGREES)
  drivetrain.drive_for(FORWARD, 18, INCHES)
  drivetrain.drive_for(REVERSE, 18, INCHES)


  # pick up and score the triball resting on the center line
  intake_motor.spin(FORWARD)
```

```
  drivetrain.turn_to_heading(140, DEGREES)
  # this part "flicks" the robot's arm, allowing it to pick up the triball with
the least amount of time spent in contact with it
  wait(0.18, SECONDS)
  drivetrain.turn_to_heading(90, DEGREES)
  intake_motor.spin(REVERSE)
  drivetrain.drive_for(FORWARD, DRIVEDIST, INCHES)
  drivetrain.drive_for(REVERSE, DRIVEDIST, INCHES)


  # pick up the triball that was above the center line triball and score that
into the net with the same flick method
  intake_motor.spin(FORWARD)
  drivetrain.turn_to_heading(30, DEGREES)
  wait(0.18, SECONDS)
  drivetrain.turn_to_heading(85, DEGREES)
  intake_motor.spin(REVERSE)
  drivetrain.drive_for(FORWARD, DRIVEDIST, INCHES)
  drivetrain.drive_for(REVERSE, DRIVEDIST, INCHES)


  # topmost center green triball - turn to grab, back up to the bottom half
where there's more room, and shoot
  intake_motor.spin(FORWARD)
  drivetrain.turn_to_heading(20, DEGREES)
  drivetrain.drive_for(FORWARD, 20, INCHES)
  drivetrain.turn_to_heading(5, DEGREES)
  drivetrain.drive_for(REVERSE, 35, INCHES)
  drivetrain.turn_to_heading(90, DEGREES)
  intake_motor.spin(REVERSE)
  drivetrain.drive_for(FORWARD, DRIVEDIST, INCHES)
  arm_motor.spin(FORWARD)
  drivetrain.drive_for(REVERSE, DRIVEDIST, INCHES)


  # turn, grab, and score the lower triball
  intake_motor.spin(FORWARD)
  drivetrain.turn_to_heading(160, DEGREES)
```

```
  drivetrain.drive_for(FORWARD, 10, INCHES)
  intake_motor.spin(REVERSE)
  drivetrain.turn_to_heading(90, DEGREES)
  drivetrain.drive_for(FORWARD, DRIVEDIST, INCHES)
  drivetrain.drive_for(REVERSE, DRIVEDIST+2, INCHES)


  #bottom line triball - the robot grabs it, crosses over into the other side,
and scores it. This gives us access to the triballs at the bottom of the field.
  intake_motor.spin(FORWARD)
  drivetrain.turn_to_heading(170, DEGREES)
  drivetrain.drive_for(FORWARD, 15, INCHES)
  drivetrain.turn_to_heading(90, DEGREES)
  drivetrain.drive_for(FORWARD, 30, INCHES)
  intake_motor.spin(REVERSE)
  drivetrain.turn_to_heading(65, DEGREES)
  drivetrain.drive_for(FORWARD, 10, INCHES)
  drivetrain.drive_for(REVERSE, 10, INCHES)


# score 2 lower triballs into the net
def bottom_triballs():
  # lower blue match load zone triball - grab it from the match load zone, then
turn and launch into the side of the goal.
  intake_motor.spin(FORWARD)
  drivetrain.turn_to_heading(132, DEGREES)
  drivetrain.drive_for(FORWARD, 40, INCHES)
  drivetrain.turn_to_heading(0, DEGREES)
  intake_motor.spin(REVERSE)
  drivetrain.drive_for(FORWARD, 10, INCHES)
  wait(0.3, SECONDS)
  drivetrain.turn_to_heading(220, DEGREES)

  # center low triball - turn and grab, then drive back and launch into the
side of the goal
  intake_motor.spin(FORWARD)
  drivetrain.drive_for(FORWARD, 20, INCHES)
  drivetrain.turn_to_heading(270, DEGREES)
```

```
  drivetrain.drive_for(FORWARD, 30, INCHES)
  drivetrain.drive_for(REVERSE, 30, INCHES)
  intake_motor.spin(REVERSE)
  drivetrain.turn_to_heading(40, DEGREES)
  drivetrain.drive_for(FORWARD, 15, INCHES)


# scores the first match loads
def first_match_load():
  # travel across the bottom, grab the triball from the match load zone, and
score the first match load by driving back up to the center
  drivetrain.drive_for(REVERSE, 14, INCHES)
  drivetrain.turn_to_heading(270, DEGREES)
  intake_motor.spin(FORWARD)
  drivetrain.drive_for(FORWARD, 94, INCHES)
  drivetrain.turn_to_heading(223, DEGREES)
  drivetrain.drive_for(FORWARD, 4, INCHES)
  wait(30, MSEC)
  drivetrain.drive_for(REVERSE, 54, INCHES)
  # We found that slightly lifting the arm increases the launch distance
  arm_motor.spin_for(REVERSE, 60, DEGREES, wait=False)
  drivetrain.drive_for(REVERSE, 10, INCHES)
  drivetrain.turn_to_heading(100, DEGREES)
  intake_motor.spin(REVERSE)
  drivetrain.turn_to_heading(90, DEGREES)
  drivetrain.drive_for(FORWARD, DRIVEDIST, INCHES)
  arm_motor.spin(FORWARD)
  drivetrain.drive_for(REVERSE, DRIVEDIST, INCHES)


#function to cycle repeated match loads via for loop
def match_loads(loadCount):
  # the for loop takes in loadCount variable provided by the function and
repeats the indented code that amount of time.
  for _ in range(loadCount):
      # this turns the robot toward the match load zone and drives forward to
grab it
      drivetrain.turn_to_heading(225, DEGREES)
```

```
      intake_motor.spin(FORWARD)
      drivetrain.drive_for(FORWARD, 64, INCHES)
      # the pause is to allow for guaranteed triball pickup
      wait(30, MSEC)
      # the robot then backs up, turns around, and shoots the triball across
the field into the net.
      drivetrain.drive_for(REVERSE, 54, INCHES)
      arm_motor.spin_for(REVERSE, 65, DEGREES, wait=False)
      drivetrain.drive_for(REVERSE, 10, INCHES)
      drivetrain.turn_to_heading(100, DEGREES)
      intake_motor.spin(REVERSE)
      drivetrain.turn_to_heading(89, DEGREES)
      drivetrain.drive_for(FORWARD, DRIVEDIST, INCHES)
      arm_motor.spin(FORWARD)
      drivetrain.drive_for(REVERSE, DRIVEDIST, INCHES)


# Define global variables
MATCHLOADS = 4 # number of match loads attempted after first
DRIVEDIST = 25 # distance driven before and after each shot to gain momentum


def main():
  #Code divided into functions for easier editng
  setup()
  preloads()
  center_triballs()
  bottom_triballs()
  first_match_load()
  match_loads(MATCHLOADS)


# VR thread - Do not delete. Found that running multiple instances of this
makes the robot wack out
vr_thread(main)
```

Side Notes: Code will only run consistently if the FPS of the simulation is kept between 59 and 63. Anything higher or lower will affect the run.