VRC 2023-2024

VEX VR Challenge – Code PDF

Gael Force Robotics

Team 5327S - Gael Force Swag

By: Atiksh Paul, Aethlyn Lim, Nico Singh, Easton Nguyen, Ashish Bhogasamudram, Lukas Somwong, Kimson Li, Iji Heo, Shrinidhi Rudrashetty, Stephanie Wang, Natalie Attalla, Rishi Shirol, Vivek Moorkoth, Phoebe Yee

Dublin High School – Dublin, CA

```python
#region VEXcode Generated Robot Configuration
import math
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:      VEX VR Online Challenge
#   Author:       5327S
#   Created:
#   Description:  VEXcode VR Python Project
#
# ----------------------------------------

# imports
import time

# initialize a class
# this will contain all of the relevant location information for the robot's movement
class RelativeLocation:

    # initialize to (0, 0, 0)
    def __init__(self):
        self.start_x = 0
        self.start_y = 0
        self.heading = 90

    # calibrates the location to the output of the GPS sensor
    def set_origin(self):
        self.start_x = gps.x_position(MM)
        self.start_y = gps.y_position(MM)
        self.start_heading = 90.0

    # calculates the change in position by subtracting starting position from current position
```

```
    def get_pos(self):
        relative_x = gps.x_position(MM) - self.start_x
        relative_y = gps.y_position(MM) - self.start_y

        return (relative_x, relative_y)

    # calculate the heading of the robot
    def get_heading(self):
        relative_heading = 90 - gps.heading()

        return relative_heading

position_tracker = RelativeLocation()

# Add project code in "main"
def main():
    drivetrain.set_drive_velocity(100, PERCENT)
    arm_motor.set_velocity(100, PERCENT)
    intake_motor.set_velocity(100, PERCENT)

    position_tracker.set_origin()

    # Drop arm and outake match load
    arm_motor.spin_to_position(1300, DEGREES, wait=False)
    lateral_pid(1400, 2)
    drivetrain.turn_for(LEFT, 45, DEGREES, wait=True)
    wait(0.1, SECONDS)
    outake()

    # Get second red ball and score
    drivetrain.turn_for(RIGHT, 45, DEGREES, wait=True)
    lateral_pid(100, 2)
    intake()
    drivetrain.turn_for(LEFT, 52, DEGREES, wait=True)
    outake()

    # Face center balls and push to other goal
    drivetrain.turn_for(RIGHT, 142, DEGREES, wait=True)
    intake()
    lateral_pid(1700, 2)
    outake()

    # Go to bottom center ball
    drivetrain.turn_for(RIGHT, 90, DEGREES, wait=True)
    lateral_pid(700, 2)
    drivetrain.turn_for(RIGHT, 90, DEGREES, wait=True)
```

```
    lateral_pid(550, 2)
    intake()

    #Score bottom center ball
    drivetrain.turn_for(LEFT, 180, DEGREES, wait=True)
    lateral_pid(560, 2)
    drivetrain.turn_for(LEFT, 90, DEGREES, wait=True)
    lateral_pid(300, 2)
    drivetrain.turn_for(RIGHT, 90, DEGREES, wait=True)
    outake()

    #Go to alley triball
    drivetrain.turn_for(RIGHT, 180, DEGREES, wait=True)
    lateral_pid(560, 2)
    drivetrain.turn_for(LEFT, 90, DEGREES, wait=True)
    lateral_pid(1150, 2)
    drivetrain.turn_for(RIGHT, 90, DEGREES, wait=True)
    intake()

    #Score Triball
    drivetrain.turn_for(RIGHT, 90, DEGREES, wait=True)
    lateral_pid(1300, 2)
    drivetrain.turn_for(RIGHT, 90, DEGREES, wait=True)
    lateral_pid(500, 2)
    outake()

    #Grab other center triball
    drivetrain.turn_for(LEFT, 90, DEGREES, wait=True)
    lateral_pid(685, 2)
    drivetrain.turn_for(LEFT, 90, DEGREES, wait=True)
    lateral_pid(580, 2)
    intake()

    #Score other center triball
    drivetrain.turn_for(LEFT, 90, DEGREES, wait=True)
    lateral_pid(410, 2)
    drivetrain.turn_for(LEFT, 90, DEGREES, wait=True)
    lateral_pid(510, 2)
    outake()

    #Get upper alley triball
    lateral_pid(500, 2, REVERSE)
    drivetrain.turn_for(LEFT, 90, DEGREES, wait=True)
    lateral_pid(1300, 2)
    drivetrain.turn_for(LEFT, 90, DEGREES, wait=True)
    intake()
```

```
    #Score upper alley triball
    drivetrain.turn_for(LEFT, 180, DEGREES, wait=True)
    outake()




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

# intake function - runs the intake motor for the specified amount of time to intake a triball
def intake():
    intake_motor.spin_for(FORWARD, 100, DEGREES)


# outtake function - runs the intake motor for the specified amount of time to outtake a triball
# utilizes the optical sensor to see when the triball is fully out of the intake
def outake():
    if optical.is_near_object():
        while optical.color() == RED.value or optical.color() == GREEN.value:
            intake_motor.spin(REVERSE)

            # delay to limit while loop memory usage
            wait(0.05, MSEC)

    intake_motor.stop()


# lateral pid loop
# PID is an algorithm used to create more consistent autonomous movements, by moving the robot to
a controlled stop.
# While this is not needed in this simulation as there is no inertia, we wanted to model a real
application that we would use.
def lateral_pid(distance, timeout=3, direction=FORWARD):
    # Tuning variables
    kP = 0.3
    kI = 0.0001
    kD = 0.09

    # # Wheel diameter: 69.85 mm
    # # Calculates circumference of wheel
    # circumference = 69.85 * math.pi

    # Declare variables
    previous_error = 0
```

```
    integral = 0
    error = 0

    # set initial coordinates
    initial_x, initial_y = position_tracker.get_pos()

    # debugging/testing
    # brain.screen.print("Heading: " + str(position_tracker.get_heading()) + "\n")

    # update heading
    current_heading = position_tracker.get_heading()

    # sets the robot's position to face the opposite direction if the turn is reversed
    if direction == REVERSE:
        current_heading += 180

    # calculate the angle needed to turn
    angle = current_heading * (math.pi / 180)

    # calculate the x and y distances from the target distance
    delta_x = distance * math.cos(angle)
    delta_y = distance * math.sin(angle)

    # final coordinates - update initial values with deltas
    final_x = initial_x + delta_x
    final_y = initial_y + delta_y

    # debugging/testing
    # brain.screen.print("angle: " + str(current_heading) + "\n")
    # brain.screen.print("finalX: " + str(final_x) + "\n")
    # brain.screen.print("finalY: " + str(final_y) + "\n")

    # Define tolerance for error
    tolerance = 5  # Adjust as needed

    start_time = time.time()

    # Main pid loop: checks error to see if bot is at destination
    while True:
        # Error resets every time to see new distance

        current_x, current_y = position_tracker.get_pos()

        # brain.screen.print("currentX: " + str(current_x) + "\n")
        # brain.screen.print("currentY: " + str(current_y) + "\n")
```

```
        error = math.sqrt(math.pow(final_x - current_x, 2) + math.pow(final_y - current_y, 2))

        # brain.screen.print("Error: " + str(error) + "\n")

        # Check if error is within tolerance
        if abs(error) <= tolerance:
            break

        # Integral accumulates error with anti-windup
        integral += error
        if integral > 100:
            integral = 100
        elif integral < -100:
            integral = -100

        # Derivative checks instantaneous slope
        derivative = error - previous_error

        # Power calculation using values and tuning variables
        # brain.screen.print("err:" + str(error * kP) + " der:" + str(derivative * kD) + " int:"
+ str(integral * kI) + "\n")
        power = error * kP + derivative * kD + integral * kI

        # Set previous error to error (used for derivative)
        previous_error = error

        # Cap power within limits
        power = max(min(power, 100), 0)

        # Uses power from pid as velocity
        # brain.screen.print("Power: " + str(power) + "\n")

        # set drivetrain to this speed
        drivetrain.set_drive_velocity(power, PERCENT)

        # move drivetrain
        drivetrain.drive(direction)

        # exit condition for loop
        if time.time() - start_time > timeout:
            drivetrain.stop()
            return

        # Delay for health reasons
        wait(0.5, MSEC)
```

```
    # Stop the drivetrain after reaching the target
    drivetrain.stop()
```