14683B Virtual Skills Code

Members: Sean, Matthew, Allen, Cal, Hiro

Team Number: 14683B

Location: Taipei, Taiwan

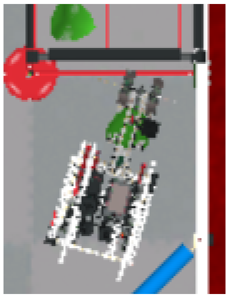
3	87	1	14683B	KCIS Robotics B

SMACH

@kcis_14683b

INDEX:

- **Two scoring methods**
  Low-score: drop the intake and score near the goal, more efficient when the robot is in red offensive zone

  High_score: rise the intake and outtake from far away. The ball will roll into the goal due to inertia

- **Brief intro of functions:**

Function Name	**What it does**
calDeg	Calculate the angle the robot should face so it can reach the goal using **arctan**
calDist	Calculate the distance between the robot and the goal using **Pythagorean theory**
cal_goal_for_angle	Calculate the where the robot should stop to allow the intake located on certain location at certain angle using **sine and cosine**
need_heading_corr	Check if the robot needs to adjust its heading to keep facing the goal using **GPS sensor**

hasball	Use **optical sensor** to check if the ball is still inside the intake
move	Make the robot face the goal, move the robot toward the goal, and return the distance between the robot and the goal after moving
goto	Make the robot moves to a location
goto_and_catch	Make the robot moves to a triball and grab it
cal_scoring_pt	Calculate a scoring point closest to the robot within a range
best_scoring_loc	Use **greedy algorithm** to get the closest scoring point among every available scoring ranges
goto_and_low_score	Move to the best scoring point and outtake the triball
low_score	Combine best_scoring_lco and goto_and_low_score
high_score	Goto scoring spot and shoot the ball

- **How we use the sensors:**
  Optical sensor: Since the Optical sensor is set right behind the intake, we use it to check if the ball is still inside the intake. It allow us to determined whether the ball has been successfully shot or caught. The use of optical sensor can make both catching and scoring process faster and more stable, enabling the robot process to the next action once the ball has left or get into the intake properly.

  GPS sensor: The use of gps sensor allow us to monitor both position and heading of the robot, facilitating the calculate the distance between the robot and the target, as well as the angle the robot should face. The process ensure the accuracy of the robot's movement moves accurately. Further more, using gps sensor also makes the code nice and clean. Instead of directly telling the robot to face which direction and go how much distance, we simply provide the robot the locations of each triballs and range of the goal, then the robot can use the data gps sensor provided to calculate the best route and move.

Motor sensing: We used motor sensing when catching the ball. In our code, the robot's arm changes angle when using high-score method. To make sure the robot's arm has been set back to the original position, the code first tell the robot to spin back to the catching position without waiting the process has done(saves time), then check if the arm is still moving after reaching the triballs' locations.

- **How we represent variables:**
  Triballs' locatoins (Triballs on the field and preloads):
  [X, Y, Angle to catch]

  Low Scoring ranges (For both preloads and green triballs):
  [[x_range, x_range], where to stop(mm from goal), where to shoot(mm from goal)]

  High Scoring spot: [X, Y, Angle]

- **Our code:**

```
#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:      VEXcode Project
#   Author:       VEX
#   Created:
#   Description:  VEXcode VR Python Project
#
# ------------------------------------------
# Add project code in "main"

#set velocity for each motors
drivetrain.set_drive_velocity(100, PERCENT)
drivetrain.set_turn_velocity(100, PERCENT)
arm_motor.set_velocity(100, PERCENT)
intake_motor.set_velocity(100, PERCENT);

#allow error when turning or traveling(impossible to be on exact number)
degError = 1.8
distError = 35

#distance between gps and intake in mm
intake_length = 295

#index of each data(gives index a meaning)
```

```python
locX = 0
locY = 1
locAng = 2
stop_point = 2
shoot_point = 3

#field preload location
preload_loc = [[-900, 300, None]]

#locations of every green triballs on the court and the angle the robot should face to grab the ball ([x, y, angle])
triballs_loc = [[-120, 0, None], [-100, -600, 270], [-100, -1050, 225], [1600, -1600, 135], [0, -1500, 245], [-125, 600, 270], [-125, 1050, 315], [0, 1500, 90], [1600, 1600, 45]]

#match load location ([x, y, angle])
loadzone = [-1600, 1600, 315]

#low_scoring_loc for preload triballs
preload_scoring_loc = [[[-1200, -1200], [400, -400], 300, 300]]

#available scoring ranges when arm is out, where should robot stop(mm from goal), and when should it release the ball(mm from goal)([[x_range, x_range], where to stop, where to shoot])
low_scoring_loc = [[[1200, 1200], [400, -400], 600, 1150], [[1300, 1600], [-600, -600], 250, 600], [[1300, 1600], [600, 600], 250, 600]]

#location and ang that allows the robot to throw the ball into the goal on blue offensive zone ([x, y, angle])
high_scoring_loc = [-30, 450, 270]

#position of the arm when doing different things
arm_catch = 1260
arm_high_score = 110

def calDeg(x_curr, y_curr, x_goal, y_goal):
   #using trigonometry to calculate the angle the robot should face
   return math.degrees(math.atan2(x_goal - x_curr, y_goal - y_curr))

def calDist(x_curr, y_curr, x_goal, y_goal):
   #using pythagorean theory to calculate the distance between two points
   return ((x_goal - x_curr)**2 + (y_goal - y_curr)**2) ** 0.5
```

```python
def cal_goal_for_ang(x_goal, y_goal, direction, ang):
   #calculate where the robot should stop so the intake will locate on the goal
when facing certain angle

   #if the robot is traveling in opposite direction, then the final angle will
also be opposite
   if (direction == REVERSE):
       ang -= 180
       if (ang < 0):
           ang += 360

   #calculate new goal
   new_x_goal = x_goal
   new_y_goal = y_goal

   if (ang != None):
       new_x_goal -= intake_length * math.cos(math.radians((450 - ang) % 360))
       new_y_goal -= intake_length * math.sin(math.radians((450 - ang) % 360))

   return new_x_goal, new_y_goal

def need_heading_corr(deg):
   #check if the robot is need to adjust its heading

   currHeading = gps.heading()
   bound_low = deg - degError
   bound_high = deg + degError

   #check if the current heading is out of acceptable range
   if (bound_low < 0 or bound_high > 360):
       if (bound_low < 0):
           bound_low += 360
       else:
           bound_high -= 360
       if (currHeading > bound_low or currHeading < bound_high):
           return False
       return True
   else:
       if (currHeading > bound_low and currHeading < bound_high):
           return False
       return True
```

```python
def hasball():
   #using optical sensor on the intake to check if the ball is still inside the intake

   if ((optical.hue() == 120 or optical.hue() == 0) and optical.is_near_object()):
       return True

   return False

def move(x_goal, y_goal, direction = FORWARD):
   #move the robot toward the goal and return distance between the robot and the goal

   x_curr = gps.x_position(MM)
   y_curr = gps.y_position(MM)

   #calculate the angle the robot should face
   deg = calDeg(x_curr, y_curr, x_goal, y_goal)

   if (direction == REVERSE):
       deg -= 180

   if (deg < 0):
       deg += 360

   #adjust the heading the robot is facing if it is not facing the goal
   if (need_heading_corr(deg)):
       drivetrain.turn_to_heading(deg, DEGREES)

   drivetrain.drive(direction)

   #calculate the distance between the robot and the goal
   dist = calDist(x_curr, y_curr, x_goal, y_goal)

   return dist

def goto(x_goal, y_goal, direction = FORWARD):
   #Let the robot move to a specified coordinates

   dist_from_goal = move(x_goal, y_goal, direction)
```

```
    while not (dist_from_goal < distError):
        dist_from_goal = move(x_goal, y_goal, direction)
        wait(5, MSEC)

def goto_and_catch(x_goal, y_goal, ang):
    #Let the robot go to the location of the ball and grab the ball

    #drop the intake to the catch position
    arm_motor.spin_to_position(arm_catch, DEGREES, wait=False)

    #calculate where should the robot reach so the intake can grab the ball at a
certain angle
    new_x_goal, new_y_goal = cal_goal_for_ang(x_goal, y_goal, FORWARD, ang)
    dist_from_goal = move(new_x_goal, new_y_goal)

    intake_motor.spin(FORWARD)

    while not (dist_from_goal < distError):
        #move toward the ball
        dist_from_goal = move(new_x_goal, new_y_goal)
        wait(5, MSEC)

    #face the catching angle
    if (ang != None):
        drivetrain.turn_to_heading(ang, DEGREES)

    #wait till the arm motor is completely set and the ball is caught by the
intake
    while(arm_motor.is_spinning() or (not hasball())):
        wait(5, MSEC)

    drivetrain.stop()

def cal_scoring_pt(x_curr, y_curr, x_range, y_range):
    #calculate the scoring position that is the closest to the robot within a
range

    if (x_range[0] == x_range[1]):
        if (y_curr < y_range[0] and y_curr > y_range[1]):
            return [x_range[0], y_curr]
        else:
```

```
            if (y_curr > y_range[0]):
                return [x_range[0], y_range[0]]
        return [x_range[0], y_range[1]]
    else:
        if (x_curr < x_range[0] and x_curr > x_range[1]):
            return [x_curr, y_range[0]]
        else:
            if (y_curr > y_range[0]):
                return [x_range[0], y_range[0]]
        return [x_range[1], y_range[0]]

def best_scoring_loc(locations):
    #choose the best scoring location amoung all available scoring ranges

    x_curr = gps.x_position(MM)
    y_curr = gps.y_position(MM)

    mmin = sys.maxsize
    index = -1
    goal = []

    for i in range(len(locations)):
        #loop through all the location and find the closest one by comparesing
the distance bewtween robot and possible goal
        locs = cal_scoring_pt(x_curr, y_curr, locations[i][0], locations[i][1])
        dist = calDist(x_curr, y_curr, locs[locX], locs[locY])

        #update the closest spot
        if (dist < mmin):
            mmin = dist
            index = i
            goal = locs

    return goal, locations[index][shoot_point], locations[index][stop_point]

def goto_and_low_score(goal, shoot_pt, stop_pt):
    #make the robot goto the closest available scoring position and shoot the
ball into the goal

    dist_from_goal = move(goal[locX], goal[locY])

    while not (dist_from_goal < distError + stop_pt):
```

```
        #move to the shooting spot
        dist_from_goal = move(goal[locX], goal[locY])

        #start shooting before reaching the goal allows the triball get further
inside the goal and avoid blocking other triballs at the goal entance
        if (dist_from_goal < distError + shoot_pt):
            intake_motor.spin(REVERSE)

        wait(5, MSEC)

    drivetrain.stop()

    #keep shooting until the ball get out of the intake
    while(hasball()):
        intake_motor.spin(REVERSE)
        wait(5, MSEC)

    intake_motor.stop()

def low_score(scoring_locs):
    #calculate the closest scroing spot and score at the location
    goal, shoot_pt, stop_pt = best_scoring_loc(scoring_locs)
    goto_and_low_score(goal, shoot_pt, stop_pt)

def high_score():
    #rise the arm to allow
    arm_motor.spin_to_position(arm_high_score, DEGREES, wait=False)

    #go to shooting spot
    new_x_goal, new_y_goal = cal_goal_for_ang(high_scoring_loc[locX],
high_scoring_loc[locY], REVERSE, high_scoring_loc[locAng])
    goto(new_x_goal, new_y_goal, direction = REVERSE)

    #face the goal
    drivetrain.turn_to_heading(high_scoring_loc[locAng], DEGREES)
    drivetrain.stop()

    #wait till the ball is shot
    while(hasball()):
        intake_motor.spin(REVERSE)
        wait(5, MSEC)
```

```
    #adjust the shooting angle and location so the ball can roll into the goal
easier
    high_scoring_loc[locAng] += 3.5
    high_scoring_loc[locY] -= 125

def main():
    arm_motor.set_position(0, DEGREES)
    arm_motor.spin_to_position(arm_catch, DEGREES, wait=False)
    goto(-900, 0)
    low_score(preload_scoring_loc)

    for i in range(7):
        goto_and_catch(triballs_loc[i][locX], triballs_loc[i][locY],
triballs_loc[i][locAng])
        low_score(low_scoring_loc)

    for i in range(5):
        goto_and_catch(loadzone[locX], loadzone[locY], ang = loadzone[locAng])
        high_score()

    goto_and_catch(loadzone[locX], loadzone[locY], ang = loadzone[locAng])
    #push the ball under the blue elevation bar to the offensive zone
    goto(700, 1500)
    low_score(low_scoring_loc)

    for i in range(8, 9):
        goto_and_catch(triballs_loc[i][locX], triballs_loc[i][locY], ang =
triballs_loc[i][locAng])
        low_score(low_scoring_loc)

    #let the robot turn away so it won't touches any ball in the end of the
match
    goto(300, 1200)
# VR threads TEST — Do not delete
vr_thread(main)
```