```
#pragma region VEXcode Generated Robot Configuration
// Make sure all required headers are included.
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <string.h>


#include "vex.h"

using namespace vex;

// Brain should be defined by default
brain Brain;


// START V5 MACROS
#define waitUntil(condition)
\
 do {
\
   wait(5, msec);
\
 } while (!(condition))

#define repeat(iterations)
\
 for (int iterator = 0; iterator < iterations; iterator++)
// END V5 MACROS


// Robot configuration code.
motor Catapult = motor(PORT8, ratio36_1, false);

controller Controller1 = controller(primary);
motor leftMotorA = motor(PORT1, ratio18_1, true);
motor leftMotorB = motor(PORT2, ratio18_1, true);
motor_group LeftDriveSmart = motor_group(leftMotorA, leftMotorB);
motor rightMotorA = motor(PORT4, ratio18_1, false);
```

```
motor rightMotorB = motor(PORT5, ratio18_1, false);
motor_group RightDriveSmart = motor_group(rightMotorA, rightMotorB);
drivetrain Drivetrain = drivetrain(LeftDriveSmart, RightDriveSmart,
319.19, 406.4, 40, mm, 1);

limit Catacheck = limit(Brain.ThreeWirePort.A);
motor IntakeMotorA = motor(PORT6, ratio6_1, false);
motor IntakeMotorB = motor(PORT7, ratio6_1, false);
motor_group Intake = motor_group(IntakeMotorA, IntakeMotorB);

motor Expansion = motor(PORT9, ratio36_1, false);



// define variable for remote controller enable/disable
bool RemoteControlCodeEnabled = true;
// define variables used for controlling motors based on controller inputs
bool Controller1LeftShoulderControlMotorsStopped = true;
bool Controller1RightShoulderControlMotorsStopped = true;
bool Controller1XBButtonsControlMotorsStopped = true;
bool DrivetrainLNeedsToBeStopped_Controller1 = true;
bool DrivetrainRNeedsToBeStopped_Controller1 = true;

// define a task that will handle monitoring inputs from Controller1
int rc_auto_loop_function_Controller1() {
 // process the controller input every 20 milliseconds
 // update the motors based on the input values
 while(true) {
   if(RemoteControlCodeEnabled) {

     // calculate the drivetrain motor velocities from the controller
joystick axies
     // left = Axis3 + Axis1
     // right = Axis3 - Axis1
     int drivetrainLeftSideSpeed = Controller1.Axis3.position() +
Controller1.Axis1.position();
     int drivetrainRightSideSpeed = Controller1.Axis3.position() -
Controller1.Axis1.position();
```

```
    // check if the value is inside of the deadband range
    if (drivetrainLeftSideSpeed < 5 && drivetrainLeftSideSpeed > -5) {
      // check if the left motor has already been stopped
      if (DrivetrainLNeedsToBeStopped_Controller1) {
        // stop the left drive motor
        LeftDriveSmart.stop();
        // tell the code that the left motor has been stopped
        DrivetrainLNeedsToBeStopped_Controller1 = false;
      }
    } else {
      // reset the toggle so that the deadband code knows to stop the
left motor nexttime the input is in the deadband range
      DrivetrainLNeedsToBeStopped_Controller1 = true;
    }
    // check if the value is inside of the deadband range
    if (drivetrainRightSideSpeed < 5 && drivetrainRightSideSpeed > -5) {
      // check if the right motor has already been stopped
      if (DrivetrainRNeedsToBeStopped_Controller1) {
        // stop the right drive motor
        RightDriveSmart.stop();
        // tell the code that the right motor has been stopped
        DrivetrainRNeedsToBeStopped_Controller1 = false;
      }
    } else {
      // reset the toggle so that the deadband code knows to stop the
right motor next time the input is in the deadband range
      DrivetrainRNeedsToBeStopped_Controller1 = true;
    }

    // only tell the left drive motor to spin if the values are not in
the deadband range
    if (DrivetrainLNeedsToBeStopped_Controller1) {
      LeftDriveSmart.setVelocity(drivetrainLeftSideSpeed, percent);
      LeftDriveSmart.spin(forward);
    }
    // only tell the right drive motor to spin if the values are not in
the deadband range
    if (DrivetrainRNeedsToBeStopped_Controller1) {
      RightDriveSmart.setVelocity(drivetrainRightSideSpeed, percent);
      RightDriveSmart.spin(forward);
```

```
    }
    // check the ButtonL1/ButtonL2 status to control Intake
    if (Controller1.ButtonL1.pressing()) {
      Intake.spin(forward);
      Controller1LeftShoulderControlMotorsStopped = false;
    } else if (Controller1.ButtonL2.pressing()) {
      Intake.spin(reverse);
      Controller1LeftShoulderControlMotorsStopped = false;
    } else if (!Controller1LeftShoulderControlMotorsStopped) {
      Intake.stop();
      // set the toggle so that we don't constantly tell the motor to
stop when the buttons are released
      Controller1LeftShoulderControlMotorsStopped = true;
    }
    // check the ButtonR1/ButtonR2 status to control Catapult
    if (Controller1.ButtonR1.pressing()) {
      Catapult.spin(forward);
      Controller1RightShoulderControlMotorsStopped = false;
    } else if (Controller1.ButtonR2.pressing()) {
      Catapult.spin(reverse);
      Controller1RightShoulderControlMotorsStopped = false;
    } else if (!Controller1RightShoulderControlMotorsStopped) {
      Catapult.stop();
      // set the toggle so that we don't constantly tell the motor to
stop when the buttons are released
      Controller1RightShoulderControlMotorsStopped = true;
    }
    // check the ButtonX/ButtonB status to control Expansion
    if (Controller1.ButtonX.pressing()) {
      Expansion.spin(forward);
      Controller1XBButtonsControlMotorsStopped = false;
    } else if (Controller1.ButtonB.pressing()) {
      Expansion.spin(reverse);
      Controller1XBButtonsControlMotorsStopped = false;
    } else if (!Controller1XBButtonsControlMotorsStopped) {
      Expansion.stop();
      // set the toggle so that we don't constantly tell the motor to
stop when the buttons are released
      Controller1XBButtonsControlMotorsStopped = true;
    }
```

```
    }
    // wait before repeating the process
    wait(20, msec);
  }
  return 0;
}

task rc_auto_loop_task_Controller1(rc_auto_loop_function_Controller1);
#pragma endregion VEXcode Generated Robot Configuration

// Include the V5 Library
#include "vex.h"
 // Allows for easier use of the VEX Library
using namespace vex;


float myVariable;

// "when started" hat block
int whenStarted1() {
 return 0;
}

//Brake function
void Brake() {
   Drivetrain.setStopping(hold);
   Drivetrain.stop();

   if(Controller1.ButtonA.pressing() == false){
     Drivetrain.setStopping(brake);

   }
 }


void torqueMode(){

 if(Controller1.ButtonX.pressing()){
  Drivetrain.setDriveVelocity(100,rpm);
  }else{
```

```
    Drivetrain.setDriveVelocity(200,rpm);
  }

}


void rollerMode(){

  if(Controller1.ButtonB.pressing()){
    Intake.setVelocity(150, rpm);
  }else{
    Intake.setVelocity(100, percent);
  }

}

void catapultcode(){
  while (Catacheck.pressing() == false) {
    Catapult.spin(forward);
  }
   if(Catacheck.pressing()){
    Catapult.stop();
  }

}




int main() {
  whenStarted1();
  Brain.Screen.print("BOT DIFF");

  //Catapult code-Pulls back the arm automatically into the loading
position(loading position determined by placement of a bumper sensor)
  Catapult.setStopping(hold);
  catapultcode();
  Expansion.setStopping(hold);
```

```
//I got bored and decided to code in a bunch of random functions for no
reason

//Brake Function Call- Honestly useless, but it makes the drive stop and
try to hold current position
 Controller1.ButtonA.pressed(Brake);
  //A function that should increase torque in the drivetrain-
//Only use when pushing another bot that is resisting strongly as
excessive torque may twist axles
 torqueMode();

 //Another function that should help spin stubborn rollers-it decreases
rpm to increase torque in the spinner
 rollerMode();
}
```