Team 14365 – Summit Silver Knights Programming Reference

Initial version November 11, 2018

Revisions:

Date	Revision #	Description of changes	Author

Overview

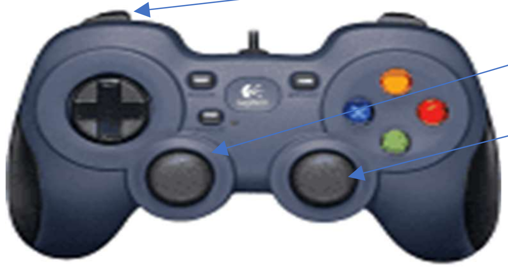
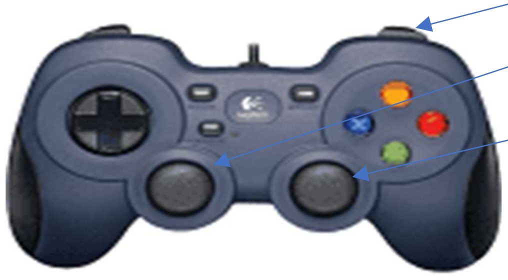
We utilize OnBot for our development. At this time we don't utilize any external libraries such as OpenCV, so we could not justify the additional complexity of Android Studio at this time.

TeleOp Mode

TeleOp mode is pretty straight-forward mapping of motors to keys. We did, however, include a Boolean (liftRaised) to ensure we don't accidentally try to raise the lift twice. We did this once by accident and broke a lift mount. The raiseLift() method will only function if liftRaised is false;

At this point we DO NOT have a working arm to gather minerals to place them in the landing craft. Instead we are depending on simply using a shovel on the rear of the robot to collect and move minerals to the claim area.

Teleop key mappings

Controller 1	
	Left_trigger (push to slow drive wheels) Left_stick_y (right Drive) Right_stick_y (Left Drive)
Controller 2	
	Right_trigger (close Lift Pin) Left_stick_y (Screw Power – left/lower Bot) Right_stick_y (Raise/lower Plow)

TeleOp Code

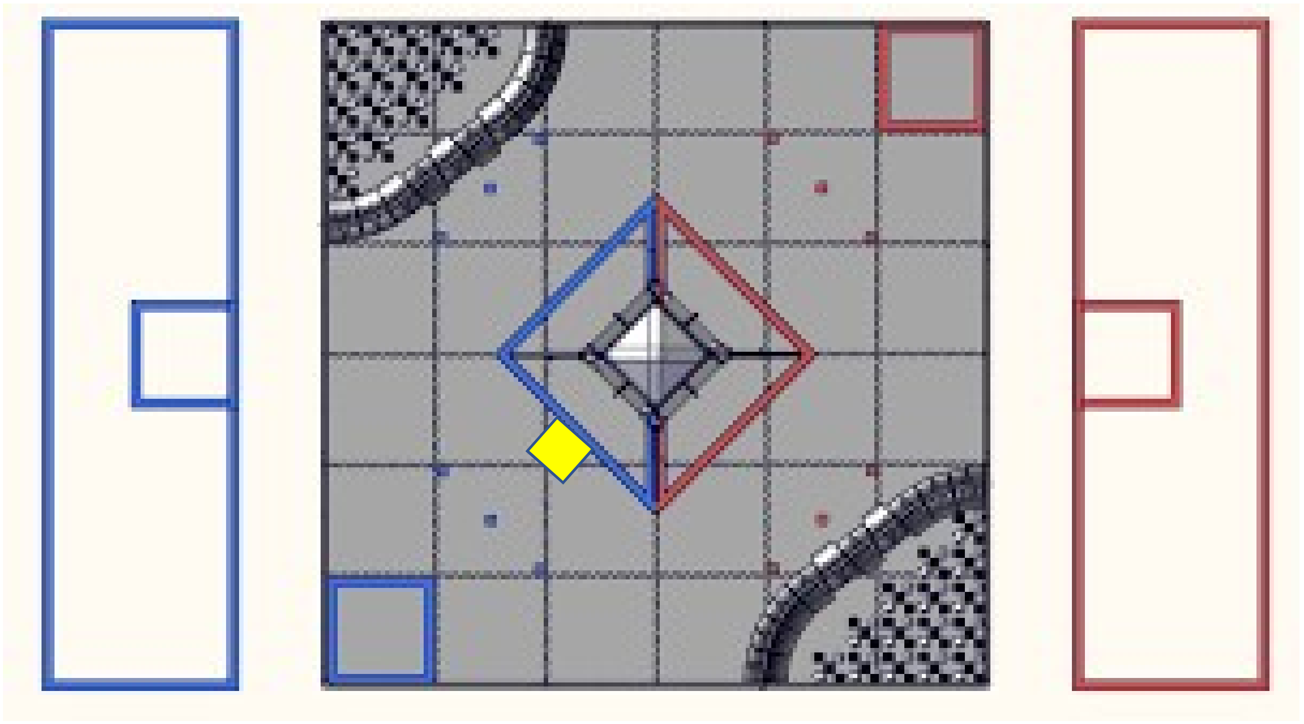
Autonomous Op Mode(s)

We have 2 unique Autonomous programs at this point. One for both potential starting positions. Our Robot starts from an attached (latched) position in both cases.

Autonomous-ClaimFacing

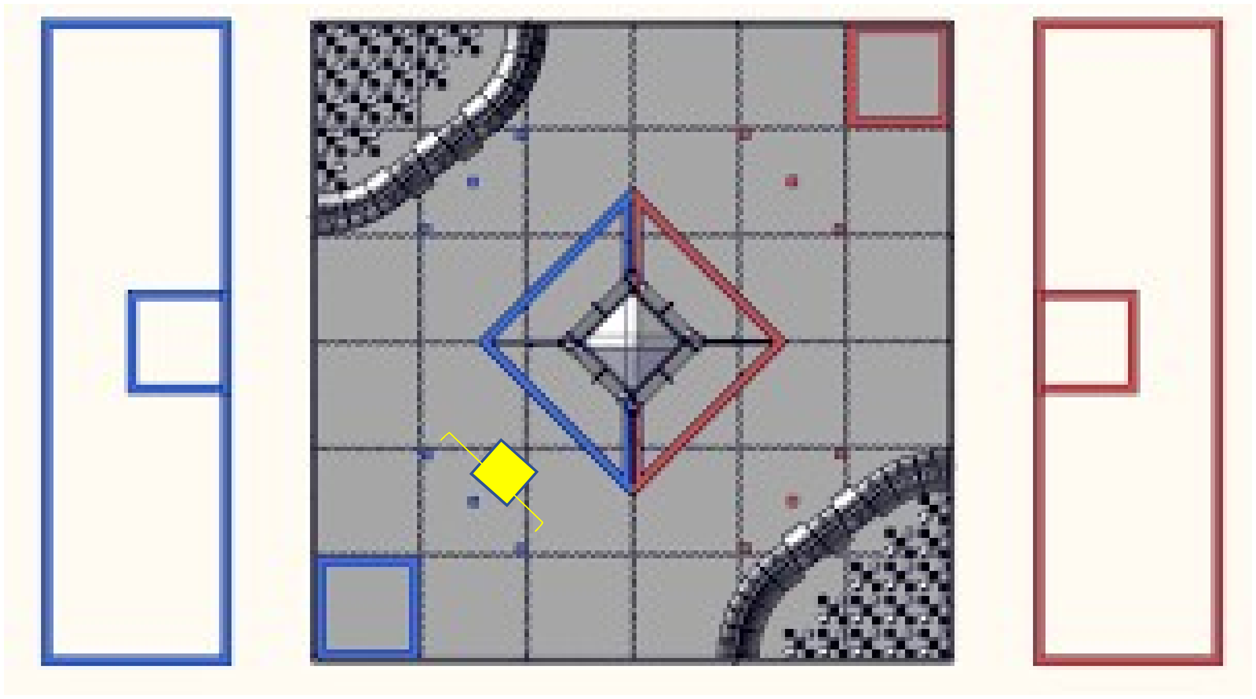
0 - Starting in attached position

Program1 – ClaimFacing1



1 – lowerRobot(); //Lowers Robot to ground + enough space to free arm

2 – moveHangArmOff (); // Moves arm sideways to complete detachment from lander



forward(N); //Move the robot forward what we see as a safe initial distance.

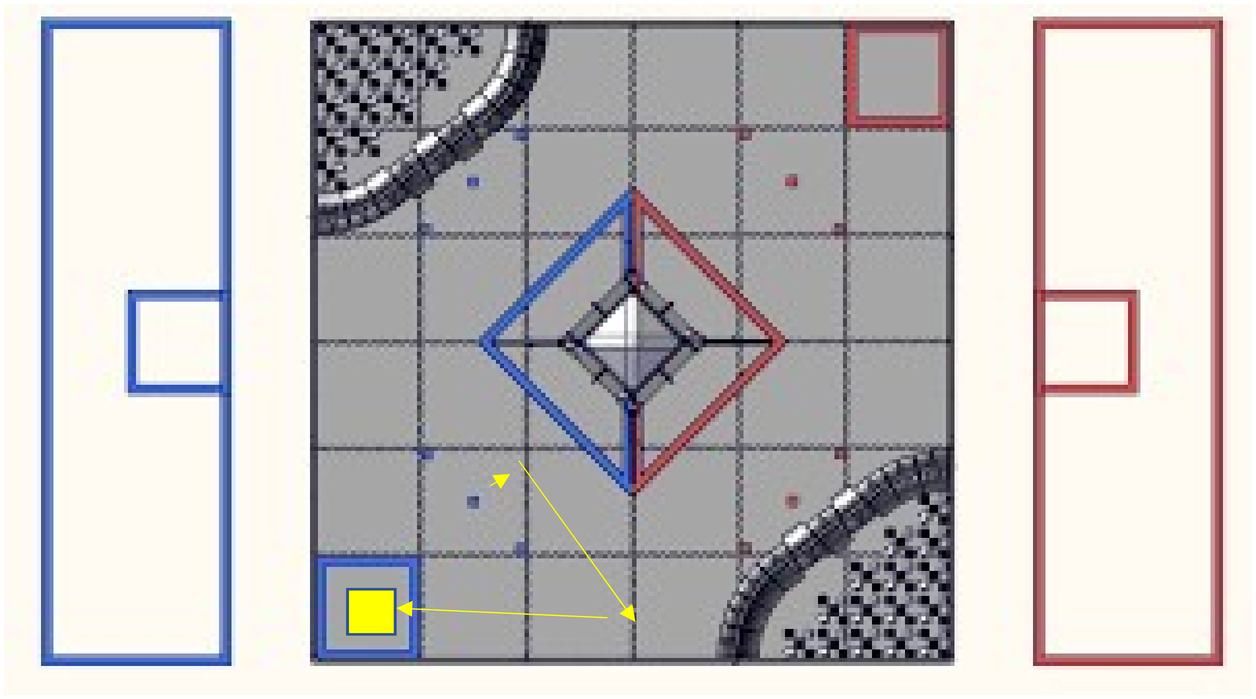
deployLeftSamplingArmToSample (); //Moves Left Sampling Arms out to prepare for sampling and distance detection

deployRightSamplingArmToSample();//Moves Right Sampling Arms out to prepare for sampling and distance

getSample(lColor,rColor) ; //get color readings of the two outside minerals to determine which of the 3 is gold

reverse(); //simply moves forward enough to “sample” the Gold Mineral. Since the robot has elevated wheels it will not hit the middle mineral unless an arm is moved to the center down position

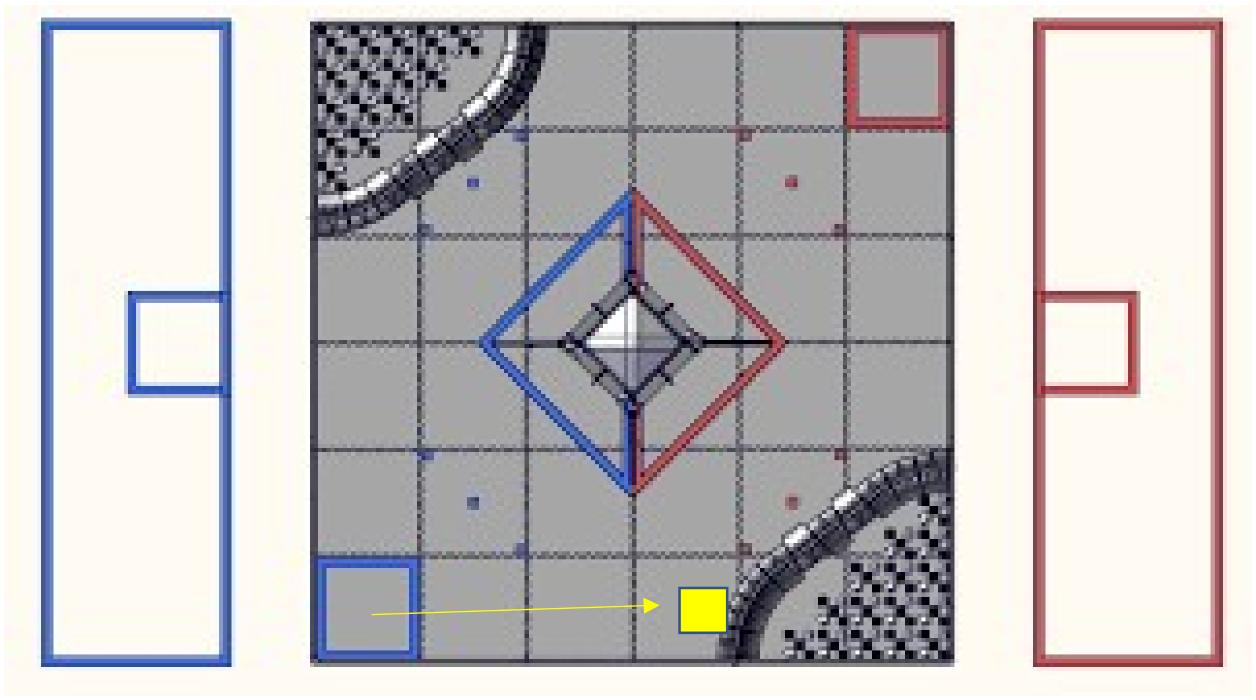
storeBothSamplingArms (); // Move both sampling arms to a safe position for remainder of game



**several turns and movements to get to claim

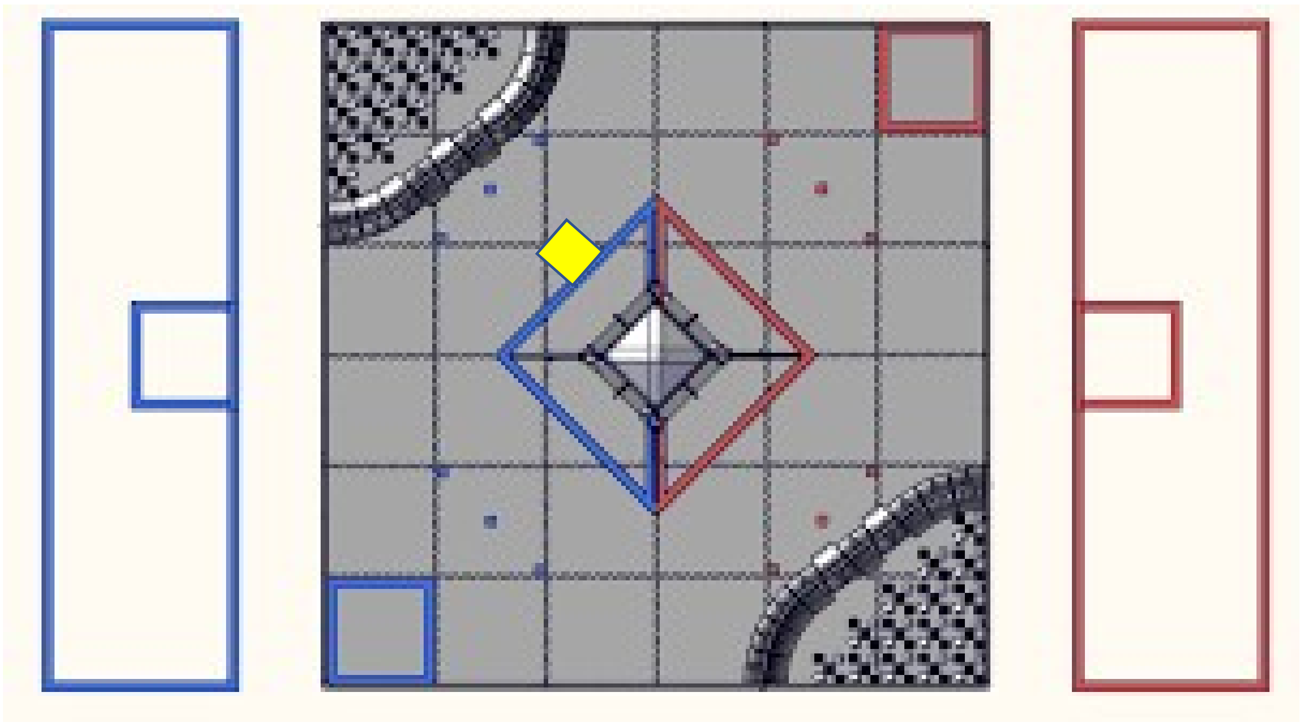
dropPlow(); //Moves Robot to corner of claim box so token can be dropped. Token is dropped lowering plow – on which token rests.   The Robot is actually "backed" into this position since the token is dropped from the back. This method is the only part that is different between the two Autonomous programs

*Future improvement is to create alternate path to claim and park to better collaborate with partner robot's autonomous program.



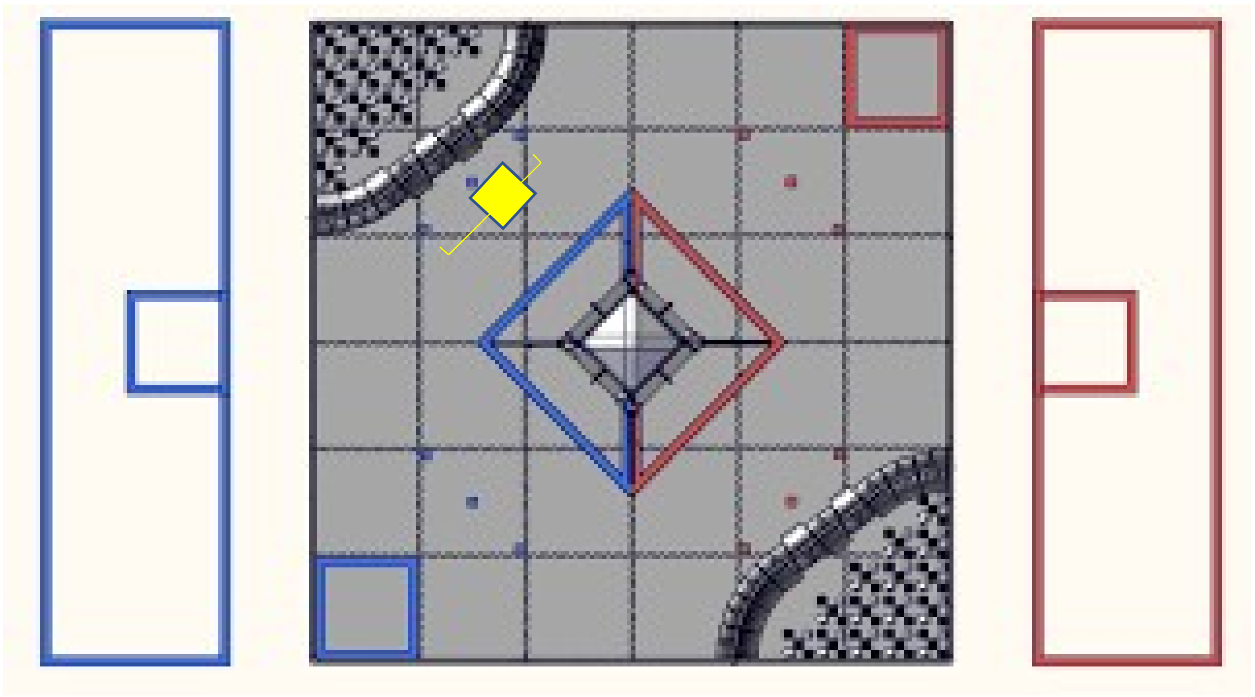
reverse();// drive forward into parked on crater

Program2 – CraterFacing1



1 – lowerRobot (); //Lowers Robot to ground + enough space to free arm

2 – moveHangArmOff (); // Moves arm sideways to complete detachment from lander



reverse(N); //Move the robot forward what we see as a safe initial distance.

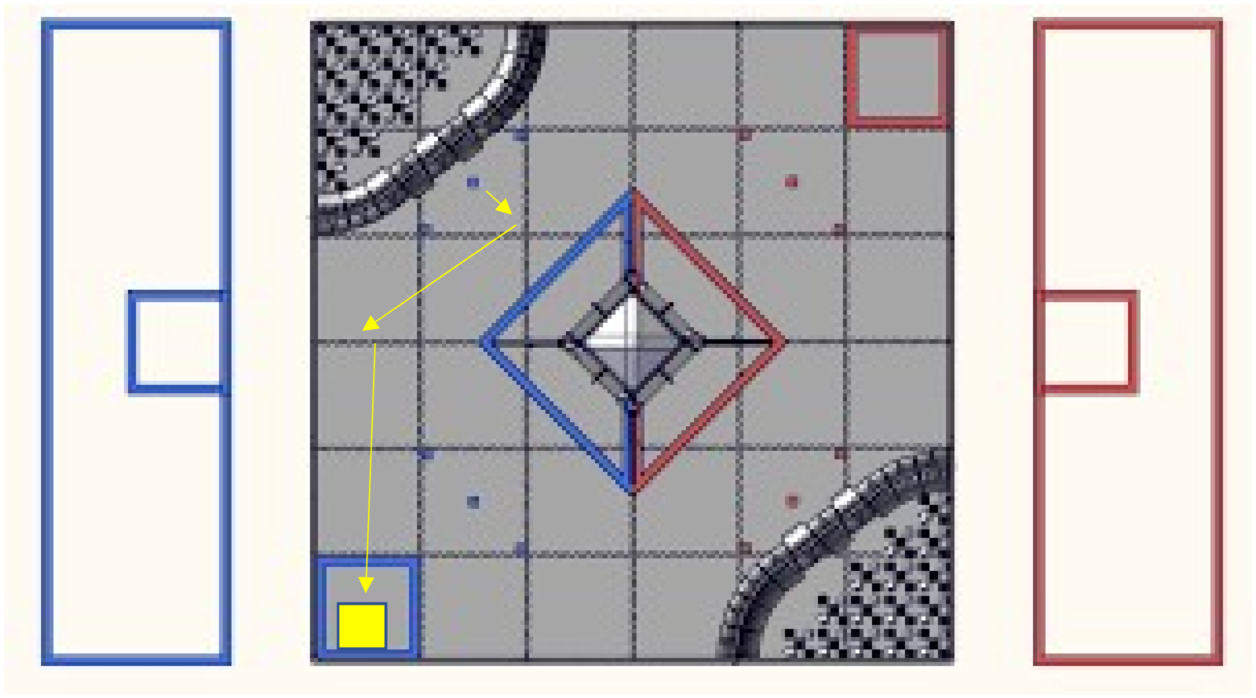
deployLeftSamplingArmToSample();

deployRightSamplingArmToSample();//Moves Right Arms out to prepare for sampling and distance detection
deployLeftSamplingArmToSample();//Moves Left Arms out to prepare for sampling and distance detection

getSample(lColor,rColor);//get color readings of the two outside minerals to determine which of the 3 is gold

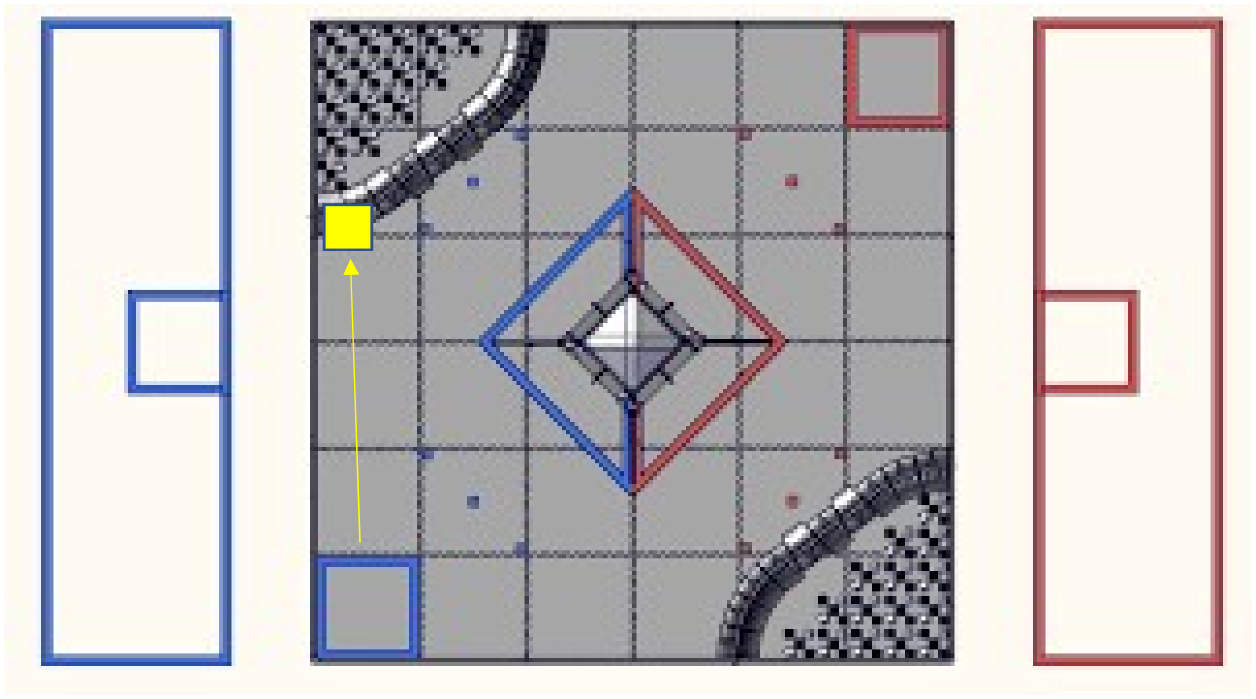
reverse(); //simply moves forward enough to “sample” the Gold Mineral. Since the robot has elevated wheels it will not hit the middle mineral unless an arm is moved to the center down position

storeBothSamplingArms (); // Move both sampling arms to a safe position for remainder of game



** various movements to get to claim

dropPlow(); //Moves Robot to corner of claim box so token can be dropped. Token is dropped lowering plow – on which the token rests. The Robot is actually “backed” into this position since the token is dropped from the back.



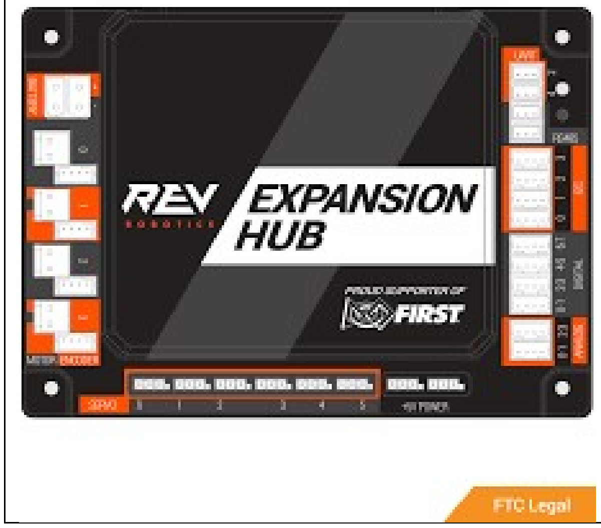
reverse();// drive forward into parked on crater

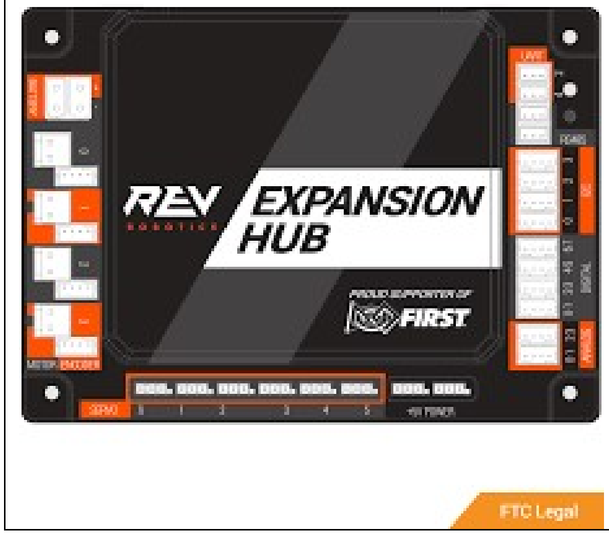
*Future improvement include:

1. Creating alternate paths to claim and park to better collaborate with partner robot's autonomous program.
2. Changing from using timed movements to using the motor encoders. While our movements work "as-is", we feel we will have even more control with encoders. This change will likely occur between our first and second tournament.
3. Move to Android Studio over onBot. OnBot has worked well and was the best tool as we started, but we would like Android Studio to improve access to outside libraries like OpenCV, and so we can more easily use tools like github to manage our code.

Sensor/Motor configuration Schematic

We utilize a 2 hub system.

Hub 1	
	<u>I2c</u> 2: rColor 3: lColor <u>Motors</u> 0: leftDrive 1: rightDrive 2: lift 3: plow <u>Servos</u> 3: liftPin 4: rightSampleArm 5: leftSampleArm

Hub 2	
	Only one hub used currrently

Code Section

TeleOp Mode

Lift

This is our one teleOp mode program to be used by drivers.

```
package com.qualcomm.ftcrobotcontroller.opmodes;


import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import java.lang.annotation.Target;
import com.qualcomm.robotcore.hardware.CRServo;
import org.firstinspires.ftc.robotcore.external.navigation.DistanceUnit;
import com.qualcomm.robotcore.eventloop.opmode.OpMode;
import org.firstinspires.ftc.robotcore.external.navigation.Rotation;
import com.qualcomm.robotcore.hardware.ColorSensor;
import com.qualcomm.robotcore.hardware.DistanceSensor;
import com.qualcomm.robotcore.eventloop.opmode.OpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.util.Range;
import com.qualcomm.robotcore.hardware.DistanceSensor;
import com.qualcomm.robotcore.hardware.Servo;

@TeleOp (name="Lift_2", group="Iterative Opmode")
public class Lift_2 extends OpMode {

   private ElapsedTime runtime = new ElapsedTime();
   private DcMotor leftDrive = null;
   private DcMotor rightDrive = null;
   private DcMotor lift = null;
   private DcMotor plow = null;
   private Servo liftPin;
   private double dpadPower = 1.0;

   double leftPower;
   double rightPower;
   double screwPower;
   double plowPower;
   double slowButton;
   double latchButton;
   @Override
   public void init() {
      telemetry.addData("Status", "Initialized");

      leftDrive  = hardwareMap.get(DcMotor.class, "leftDrive");
      rightDrive = hardwareMap.get(DcMotor.class, "rightDrive");
      lift = hardwareMap.get(DcMotor.class, "lift");
      plow = hardwareMap.get(DcMotor.class, "plow");
      liftPin = hardwareMap.get(Servo.class,"liftPin");
      //** set the direction of our motors
      leftDrive.setDirection(DcMotor.Direction.REVERSE);
      rightDrive.setDirection(DcMotor.Direction.FORWARD);
      lift.setDirection(DcMotor.Direction.FORWARD);


      telemetry.addData("Status", "Initialized");
      telemetry.update();
```

```
    }

    @Override
    public void start() {
    }

    @Override
    public void loop() {

        //** mapping the controls to certain values that will be
        //** used to control motors and servos
        //** Wheel motors
        leftPower = gamepad1.right_stick_y;
        rightPower = gamepad1.left_stick_y;
        //** Lift mechanism
        screwPower = gamepad2.left_stick_y;
        //*** Plow lower/raise
        plowPower = gamepad2.right_stick_y;

        slowButton = gamepad1.left_trigger;
        latchButton = gamepad2.right_trigger;



        double lpin = liftPin.getPosition();
        telemetry.addData("lpin",lpin);

        //** left trigger is a "slow mode" for driving
        if (slowButton==1) {
            dpadPower = 0.5;
        } else {
            dpadPower = 1.0;
        }
        //** wanted to slow the plow a little bit
        plow.setPower(plowPower*.3);

        //** Control the latching pin once ready to raise
        if (latchButton==1) {
            liftPin.setPosition(0.3);
        } else {
            liftPin.setPosition(0.88);
        }

        leftDrive.setPower(leftPower * dpadPower);
        rightDrive.setPower(rightPower  * dpadPower);

        lift.setPower(-screwPower);

        telemetry.addData("servo",liftPin.getPosition());
        telemetry.addData("plow",plow.getCurrentPosition());

    }

    @Override
    public void stop() {

    }
}
```

Autonomous Mode

AutoCraterFacing1

```
/* Copyright (c) 2017 FIRST. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted (subject to the limitations in the disclaimer below) provided that
 * the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this list
 * of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice, this
 * list of conditions and the following disclaimer in the documentation and/or
 * other materials provided with the distribution.
 *
 * Neither the name of FIRST nor the names of its contributors may be used to endorse or
 * promote products derived from this software without specific prior written permission.
 *
 * NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED BY THIS
 * LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.hardware.Gyroscope;
import com.qualcomm.robotcore.hardware.Blinker;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import java.util.concurrent.TimeUnit;
import org.firstinspires.ftc.robotcore.external.navigation.DistanceUnit;
import org.firstinspires.ftc.robotcore.external.navigation.Rotation;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DistanceSensor;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.util.Range;
import com.qualcomm.robotcore.hardware.ColorSensor;
import com.qualcomm.robotcore.eventloop.opmode.OpMode;
import com.qualcomm.robotcore.eventloop.opmode.OpMode;



/**
 * This file contains an minimal example of a Linear "OpMode". An OpMode is a 'program' that runs in either
 * the autonomous or the teleop period of an FTC match. The names of OpModes appear on the menu
 * of the FTC Driver Station. When an selection is made from the menu, the corresponding OpMode
 * class is instantiated on the Robot Controller and executed.
 *
 * This particular OpMode just executes a basic Tank Drive Teleop for a two wheeled robot
 * It includes all the skeletal structure that all linear OpModes contain.
 *
 * Use Android Studios to Copy this Class, and Paste it into your team's code folder with a new name.
 * Remove or comment out the @Disabled line to add this opmode to the Driver Station OpMode list
 */
@Autonomous(name="AutoCraterFacing1", group="Linear Opmode")

public class AutoCraterFacing1 extends LinearOpMode {

    // Declare OpMode members.
    private ElapsedTime runtime = new ElapsedTime();
    private DcMotor leftDrive = null;
    private DcMotor rightDrive = null;

    private DcMotor lift = null;
    private Servo liftPin;

    private DcMotor plow = null;

    private Servo rightSampleArm;
    private Servo leftSampleArm;
    //private Gyroscope imu;

    //The two distance sensors on the sampling arms
    private DistanceSensor lDistance;
    private DistanceSensor rDistance;
    //The two Color sensors on the sampling arms
    private ColorSensor lColor;
    private ColorSensor rColor;

    @Override
    public void runOpMode() {
        telemetry.addData("Status", "Initialized");
        telemetry.update();

        // Initialize the hardware variables. Note that the strings used here as parameters
        // to 'get' must correspond to the names assigned during the robot configuration
        // step (using the FTC Robot Controller app on the phone).
        rightDrive  = hardwareMap.get(DcMotor.class, "leftDrive");
        leftDrive = hardwareMap.get(DcMotor.class, "rightDrive");
        // Most robots need the motor on one side to be reversed to drive forward
        // Reverse the motor that runs backwards when connected directly to the battery
        leftDrive.setDirection(DcMotor.Direction.REVERSE);
        rightDrive.setDirection(DcMotor.Direction.FORWARD);
        lift = hardwareMap.get(DcMotor.class, "lift");
        plow = hardwareMap.get(DcMotor.class, "plow");
        rightSampleArm = hardwareMap.get(Servo.class,"rightSampleArm");
        leftSampleArm = hardwareMap.get(Servo.class,"leftSampleArm");
        liftPin = hardwareMap.get(Servo.class,"liftPin");

        lColor = hardwareMap.colorSensor.get("lColor");
        lDistance = hardwareMap.get(DistanceSensor.class, "lColor");
        lColor.enableLed(true);

        rColor = hardwareMap.colorSensor.get("rColor");
        rDistance = hardwareMap.get(DistanceSensor.class, "rColor");
        rColor.enableLed(true);

        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        runtime.reset();

        lowerRobot(7.0);

        moveHangArmOff();

        reverse(0.95);//was 1.0

        deployLeftSamplingArmToSample();
        deployRightSamplingArmToSample();

        pause(1); //** KEEP!! - seems to be importqant to get good color reading
        String SamplePos = getSample(lColor,rColor);
         if (SamplePos.equals("Left")){
            goldLeft(); //** move right arm out of the way
         } else if (SamplePos.equals("Right")) {
            goldRight(); //** move left arm out of the way
         } else {
            goldMiddle();

         }

        //** Move sample
        reverse(.5);

        storeBothSamplingArms(); //Just to be sure they are stored and safe

        //** Sampling complete - move to place token
        //** back toward lander
        forward(1.05);//was.85

        //** hitting an element - change angle to avoid
        rightTurn(.8);

        forward(2.25);//was 2.35

        leftTurn(.4);//

        forward(2.8);//
        dropPlow();

        reverse(2);

        //*** Token should be dropped now
        leftTurn(.05); //* corrective move
        reverse(2.1); //* forward to crater park

        stop();

    }

    //** potenial to replace the time delay used to run motors - experimental
    public void pause(double secs){
        ElapsedTime mRuntime = new ElapsedTime();
        while(mRuntime.time()< secs){

        }
    }
    //** move robot forward secs number of seconds
```

```
public void forward(double secs){

    ElapsedTime mRuntime = new ElapsedTime();
    mRuntime.reset();
    double leftPower = .75;
    double rightPower = .75;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
    while(mRuntime.time()< secs){

    }
    leftPower = 0;
    rightPower = 0;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
}
//** use encoder instead of time on motors
//** We still need to come up with a method to convert distance to pos
public void forwardEncoder(int pos){

    leftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    rightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

    leftDrive.setTargetPosition(pos);
    rightDrive.setTargetPosition(pos);
    leftDrive.setPower(0.5);
    rightDrive.setPower(0.5);
    // (pos);
}
//** move robot backward secs number of seconds
public void reverse(double secs){

    ElapsedTime mRuntime = new ElapsedTime();
    mRuntime.reset();
    double leftPower = -.75;
    double rightPower = -.75;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
    while(opModeIsActive() && mRuntime.time()< secs){

    }
    leftPower = 0;
    rightPower = 0;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);

}
//** turn robot left secs number of seconds
public void leftTurn(double secs){

    ElapsedTime mRuntime = new ElapsedTime();
    mRuntime.reset();
    double leftPower = -0.75;
    double rightPower = 0.75;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
    while(mRuntime.time()< secs){

    }
    leftPower = 0;
    rightPower = 0;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
}
    //** Turn robot Right secs number of seconds
public void rightTurn(double secs){

    ElapsedTime mRuntime = new ElapsedTime();
    mRuntime.reset();
    double leftPower = 0.75;
    double rightPower = -0.75;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
    while(mRuntime.time()< secs){

    }
    leftPower = 0;
    rightPower = 0;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
}
//*********************************************************************
//** This method RAISES the robot
//** NOTE: this means LOWERING the arm - not currently used in Autonomous
//*********************************************************************
public void raiseRobot(double secs){
    ElapsedTime mRuntime = new ElapsedTime();
    lift.setPower(-1);

    while(mRuntime.time()< secs){

    }
    lift.setPower(0);

}
//*********************************************************************
//** This method LOWERS the robot
//** NOTE: this means RAISING the arm
//*********************************************************************
public void lowerRobot(double secs){
    ElapsedTime mRuntime = new ElapsedTime();
    lift.setPower(1);

    while(mRuntime.time()< secs){

    }
    lift.setPower(0);

}
//** Moves sampling arms to correct position if Gold detected on Left
public void goldLeft(){
    deployLeftSamplingArmToSample();
    storeRightSamplingArm();
}
//** Moves sampling arms to correct position if Gold detected on Right
public void goldRight(){
    deployRightSamplingArmToSample();
    storeLeftSamplingArm();
}
//** Moves sampling arms to correct position if Gold detected on Middle
public void goldMiddle(){
    deployLeftSamplingArmToMiddle();
    pause(1);
    storeRightSamplingArm();
}
//** Detects which position we think the Gold Sample sits on
public String getSample(ColorSensor leftColor, ColorSensor rightColor) {
//***  This routine is our algorithm to determine if the color sensor in
//***  sensing the gold in the left, right or center position
    int lRed=0;
    int lGreen=0;
    int lBlue=0;
    int rRed=0;
    int rGreen=0;
    int rBlue=0;
    String goldPosition = "Left";

    lRed = leftColor.red();
    lBlue = leftColor.blue();
    lGreen = leftColor.green();
    //alpha = lColor.alpha();
    rRed = rightColor.red();
    rBlue = rightColor.blue();
    rGreen = rightColor.green();
    double lRatio = (double)lBlue/(double)lRed;
    double rRatio = (double)rBlue/(double)rRed;

    //** This is a very basic algorithm to determine "gold"
    //** We need to make this better
    if (lRatio < 0.8 && rRatio > 0.8){
        goldPosition = "Right";
    } else if (rRatio < 0.8 && lRatio > 0.8) {
        goldPosition = "Left";
    } else if (rRatio < 0.8 && rRatio < 0.8) {
        goldPosition = "Middle";
    } else if (rRatio > 0.8 && lRatio > 0.8) {
        goldPosition = "Middle";
    } else {
        goldPosition = "Middle";
    }
    telemetry.addData("left red (%)", lRed);
    telemetry.addData("left blue (%)", lBlue);
    telemetry.addData("left green (%)", lGreen);
    telemetry.addData("right red (%)", rRed);
    telemetry.addData("right blue (%)", rBlue);
    telemetry.addData("right green (%)", rGreen);
    telemetry.addData("Gold Position (%)", goldPosition);
    telemetry.addData("rRatio (%)", rRatio);
    telemetry.addData("lRatio (%)", lRatio);

    //telemetry.addData("alpha (%)", alpha);
    telemetry.update();

    return goldPosition;
}
//** Moves latching pin off so Robot can move from Lander
public void moveHangArmOff(){
    liftPin.setPosition(0.88);
    pause(1);
    //** Should only be called once the robot is dropped
}
//** Moves latching pin on so Robot can be Lifted - not used in Autonomous
public void moveHangArmOn(){
    liftPin.setPosition(0.3);
    pause(2);
    //** Should only be called once the robot in positio to be lifted
}
//** This method is intended to move the robot a close as possible to the samples
//** without toucing them.  The idea is we will move the robot a safe distance
//** close, but then we want to creep closer until the range sensor can be useful
//** NOT CURRENTLY USED
public void creepForward(DistanceSensor inDistance){

    double centerDistance = lDistance.getDistance(DistanceUnit.CM);
    Double dist = centerDistance ;
    int MAXCREEP = 50000; //** a safety in case our detectors don't work - will only let us creep forward so ClassFormatError
    int creepCount = 0;
    telemetry.addData("distance CM ", dist);
    telemetry.update();
    if (dist.isNaN()) dist = 100.0;
    while( dist > 5 && creepCount < MAXCREEP){
        forward(.3);
        pause(50);
        centerDistance = lDistance.getDistance(DistanceUnit.CM);
        dist = centerDistance ;
        if (dist.isNaN()) dist = 100.0;
        telemetry.addData("distance CM ", dist);
        telemetry.update();
        //creepCount++;
    }
}
//** moves Left sampling arm to position for sampling
public void deployLeftSamplingArmToSample(){
    leftSampleArm.setPosition(0.88); //Position to prepare left arm to sample
}
//** moves Right sampling arm to position for sampling
public void deployRightSamplingArmToSample(){
    rightSampleArm.setPosition(0.05); //Position to prepare right  arm to sample
}
//** moves Left sampling arm to position to PUSH middle element
public void deployLeftSamplingArmToMiddle(){
    leftSampleArm.setPosition(0.02); //Position to prepare left arm to sample
}
//** Stores both sampling arms back out of the way once sampling is complete
public void storeBothSamplingArms(){
    pause(1);//NEEDED?  TEST
    storeRightSamplingArm();
    storeLeftSamplingArm();
}
//** Stores Right sampling arm back out of the way
public void storeRightSamplingArm(){
    rightSampleArm.setPosition(0.6); //Right Arm Stored position
}
//** Stores Left sampling arm back out of the way
public void storeLeftSamplingArm() {
    leftSampleArm.setPosition(0.2); //Left Arm Stored position
}
//** Moves plow down so token can be deployed
public void dropPlow(){
    plow.setPower(.3);
    pause(1);
    plow.setPower(0);
}
```

```
}
```

AutoClaimFacing1

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.robot.Robot;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.hardware.Gyroscope;
import com.qualcomm.robotcore.hardware.Blinker;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import java.util.concurrent.TimeUnit;
import org.firstinspires.ftc.robotcore.external.navigation.DistanceUnit;
import org.firstinspires.ftc.robotcore.external.navigation.Rotation;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DistanceSensor;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.util.Range;
import com.qualcomm.robotcore.hardware.ColorSensor;
import com.qualcomm.robotcore.eventloop.opmode.OpMode;
import com.qualcomm.robotcore.eventloop.opmode.OpMode;



/**
 * This file contains an minimal example of a Linear "OpMode". An OpMode is a 'program' that runs in either
 * the autonomous or the teleop period of an FTC match. The names of OpModes appear on the menu
 * of the FTC Driver Station. When an selection is made from the menu, the corresponding OpMode
 * class is instantiated on the Robot Controller and executed.
 *
 * This particular OpMode just executes a basic Tank Drive Teleop for a two wheeled robot
 * It includes all the skeletal structure that all linear OpModes contain.
 *
 * Use Android Studios to Copy this Class, and Paste it into your team's code folder with a new name.
 * Remove or comment out the @Disabled line to add this opmode to the Driver Station OpMode list
 */
@Autonomous(name="AutoClaimFacing1", group="Linear Opmode")

public class AutoClaimFacing1 extends LinearOpMode {

    // Declare OpMode members.
    //counter to help keep runtime
    private ElapsedTime runtime = new ElapsedTime();
    //** our wheel motors
    private DcMotor leftDrive = null;
    private DcMotor rightDrive = null;
    //**The lift motor and the lift "latch"
    private DcMotor lift = null;
    private Servo liftPin;
    //**The plow motor
    private DcMotor plow = null;
    //** servos to move sampling arms
    private Servo rightSampleArm;
    private Servo leftSampleArm;

    //The two distance sensors on the sampling arms - not currently used
    private DistanceSensor lDistance;
    private DistanceSensor rDistance;
    //The two Color sensors on the sampling arms
    private ColorSensor lColor;
    private ColorSensor rColor;

    @Override
    public void runOpMode() {
        telemetry.addData("Status", "Initialized");
        telemetry.update();

        //*************************************************************************
        //** ultimately want to move all this setup to a seprate class called robot
        //** Currently getting cannot find symbol:   variable hardwareMap
        //** Robot robot = new Robot();
        //*************************************************************************

        // Initialize the hardware variables. Note that the strings used here as parameters
        // to 'get' must correspond to the names assigned during the robot configuration
        // step (using the FTC Robot Controller app on the phone).
        rightDrive  = hardwareMap.get(DcMotor.class, "leftDrive");
        leftDrive = hardwareMap.get(DcMotor.class, "rightDrive");
        // Most robots need the motor on one side to be reversed to drive forward
        // Reverse the motor that runs backwards when connected directly to the battery
        leftDrive.setDirection(DcMotor.Direction.REVERSE);
        rightDrive.setDirection(DcMotor.Direction.FORWARD);

        lift = hardwareMap.get(DcMotor.class, "lift");
        plow = hardwareMap.get(DcMotor.class, "plow");

        rightSampleArm = hardwareMap.get(Servo.class,"rightSampleArm");
        leftSampleArm = hardwareMap.get(Servo.class,"leftSampleArm");
        liftPin = hardwareMap.get(Servo.class,"liftPin");

        lColor = hardwareMap.colorSensor.get("lColor");
        lDistance = hardwareMap.get(DistanceSensor.class, "lColor");
        lColor.enableLed(true);
```

```
rColor = hardwareMap.colorSensor.get("rColor");
rDistance = hardwareMap.get(DistanceSensor.class, "rColor");
rColor.enableLed(true);


// Wait for the game to start (driver presses PLAY)
waitForStart();

runtime.reset();

//*Begin Autonomous
//** We think of our robot FRONT as where the plow is - so intially
//** we are facing backward
//** in most cases the paramters passed to our methods simply tell the
//** motors how many seconds to run
//*Step #1 - Lower Robot
lowerRobot(7);
//*Step #2 - detach hanging pin
moveHangArmOff();
//*Step #3 - Move forward to sample
reverse(1.1);
//*Step #4 - Deploy both sampling arms to prepare for sampling
deployLeftSamplingArmToSample();
deployRightSamplingArmToSample();

pause(1); //* pause to provide enough time for sampling arms to move
//**Step #5 - sample using color sensors move arms to appropriate spots
String SamplePos = getSample(lColor,rColor);
 if (SamplePos.equals("Left")){
    goldLeft(); //** move right arm out of the way
 } else if (SamplePos.equals("Right")) {
    goldRight(); //** move left arm out of the way
 } else {
    goldMiddle();
 }
//**Step #7 - push sample off mark
reverse(.35);
//**Step #7 - Store sampling arms
storeBothSamplingArms(); //Just to be sure they are stored and safe

//** Sampling complete - move to place token

//**Step #8 - Move back form samples
forward(0.8);
//**Step #9 - turn toward competitor crater
rightTurn(.85); // was .7

//**Step #10 - move toward competitor crater
forward(2.3);// was 2.5
//**Step #11 - turn to have token deploy facing the depot


rightTurn(.95);

//**Step #12 - Initial move toward depot
forward(1.4);//was 1.5
//**Step #13 - Adjust to approach depot aligned with wall
rightTurn(0.10);//was .12
//**Step #14 - Finish approach to depot
forward(1.5);
```

```
    //**Step #15 - Drop token
    dropPlow();
    //**Step #16 - Park in competitors crater

    reverse(3.5);
    rightTurn(.10);//was left1.0
    reverse(0.75);
    //** this code is simply used during testing to quickly comment out entire
    //** sections of code
    //if(false) {
    //}
    stop();
}
//** experimental - testing to see if this can rplace the delay code in each
//** method - not currently used
public void pause(double secs){
    ElapsedTime mRuntime = new ElapsedTime();
    while(mRuntime.time()< secs){

    }
}
//** move robot forward for secs number of seconds
public void forward(double secs){

    ElapsedTime mRuntime = new ElapsedTime();
    mRuntime.reset();
    double leftPower = .75;
    double rightPower = .75;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
    while(mRuntime.time()< secs){

    }
    leftPower = 0;
    rightPower = 0;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
}
//** Experimental - will use to replace time with encoder
//** NOT currently used
public void forwardEncoder(int pos){
//** use encoder instead of time on motors
//** We still need to come up with a method to convert distance to pos
    leftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    rightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

    leftDrive.setTargetPosition(pos);
    rightDrive.setTargetPosition(pos);
    leftDrive.setPower(0.5);
    rightDrive.setPower(0.5);
    // (pos);
}
//** Reverse Robot for secs number of seconds
public void reverse(double secs){

    ElapsedTime mRuntime = new ElapsedTime();
    mRuntime.reset();
    double leftPower = -.75;
    double rightPower = -.75;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
    while(opModeIsActive() && mRuntime.time()< secs){

    }
    leftPower = 0;
    rightPower = 0;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);

}
//** turn robot left secs number of seconds
public void leftTurn(double secs){
    ElapsedTime mRuntime = new ElapsedTime();
    mRuntime.reset();
    double leftPower = -0.75;
    double rightPower = 0.75;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
    while(mRuntime.time()< secs){

    }
    leftPower = 0;
    rightPower = 0;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
}
//** turn robot right secs number of seconds
public void rightTurn(double secs){

    ElapsedTime mRuntime = new ElapsedTime();
    mRuntime.reset();
    double leftPower = 0.75;
    double rightPower = -0.75;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
    while(mRuntime.time()< secs){

    }
    leftPower = 0;
    rightPower = 0;
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
}
//** raise robot ( by lowering the lift) secs nubmer of seconds
//** NOT currerntly used in Autonomous
public void raiseRobot(double secs){
    ElapsedTime mRuntime = new ElapsedTime();
    lift.setPower(-1);

    while(mRuntime.time()< secs){

    }
    lift.setPower(0);

}
//*********************************************************************
//** This method LOWERS the robot
//** NOTE: this means RAISING the arm
//*********************************************************************
public void lowerRobot(double secs){
    ElapsedTime mRuntime = new ElapsedTime();
    lift.setPower(1);
    while(mRuntime.time()< secs){

    }
    lift.setPower(0);

}
//** method to move sampling arms in response to gold being detected in left
//** position
public void goldLeft(){
    deployLeftSamplingArmToSample();
    storeRightSamplingArm();

}
//** method to move sampling arms in response to gold being detected in right
//** position
public void goldRight(){
    deployRightSamplingArmToSample();
    storeLeftSamplingArm();
}
//** method to move sampling arms in response to gold being detected in middle
//** position
public void goldMiddle(){
    deployLeftSamplingArmToMiddle();
    storeRightSamplingArm();
}
//** method takes in the color values of both the left and right sampling arms
//** The method then uses a ratio of blue to red values on each to determine
//** if it detects gold or silver.  Depending on what it finds it returns
//** either "Right", "Left", or "Middle" - these are then used to position
//* the sampling arms for the push step
public String getSample(ColorSensor leftColor, ColorSensor rightColor) {
    //***  This routine is our algorithm to determine if the color sensor in
    //***  sensing the gold in the left, right or center position
    int lRed=0;
    int lGreen=0;
    int lBlue=0;
    int rRed=0;
    int rGreen=0;
    int rBlue=0;
    String goldPosition = "Left";

    lRed = leftColor.red();
    lBlue = leftColor.blue();
    lGreen = leftColor.green();
    //alpha = lColor.alpha();
    rRed = rightColor.red();
    rBlue = rightColor.blue();
    rGreen = rightColor.green();
    double lRatio = (double)lBlue/(double)lRed;
    double rRatio = (double)rBlue/(double)rRed;

    //** This is a very basic algorithm to determine "gold"
    //** We need to make this better
    if (lRatio < 0.8 && rRatio > 0.8){
        goldPosition = "Right";
    } else if (rRatio < 0.8 && lRatio > 0.8) {
        goldPosition = "Left";
    } else if (rRatio < 0.8 && rRatio < 0.8) {
        goldPosition = "Middle";
    } else if (rRatio > 0.8 && lRatio > 0.8) {
        goldPosition = "Middle";
    } else {
        goldPosition = "Middle";
    }
    telemetry.addData("left red (%)", lRed);
    telemetry.addData("left blue (%)", lBlue);
    telemetry.addData("left green (%)", lGreen);
    telemetry.addData("right red (%)", rRed);
    telemetry.addData("right blue (%)", rBlue);
    telemetry.addData("right green (%)", rGreen);
    telemetry.addData("Gold Position (%)", goldPosition);
    telemetry.addData("rRatio (%)", rRatio);
    telemetry.addData("lRatio (%)", lRatio);

    telemetry.update();

    return goldPosition;
    }

//** Should only be called once the robot is dropped
public void moveHangArmOff(){
    liftPin.setPosition(0.88);
    pause(1); //keep this so we ensure robot does not move until pin is moved
}
//** should only be used in telOp - but here for consistency
public void moveHangArmOn(){
    liftPin.setPosition(0.3);
    pause(2);
}
//** This method is intended to move the robot a close as possible to the samples
//** without toucing them.  The idea is we will move the robot a safe distance
//** close, but then we want to creep closer until the range sensor can be useful
//** It is currently not used and doesn't seem to be needed - but keep in just in case
public void creepForward(DistanceSensor inDistance){

    double centerDistance = lDistance.getDistance(DistanceUnit.CM);
    Double dist = centerDistance ;
    int MAXCREEP = 50000; //** a safety in case our detectors don't work - will only let us creep forward so ClassFormatError
    int creepCount = 0;
    telemetry.addData("distance CM ", dist);
    telemetry.update();
    if (dist.isNaN()) dist = 100.0;
    while( dist > 5 && creepCount < MAXCREEP){
        forward(.3);
        pause(50);
        centerDistance = lDistance.getDistance(DistanceUnit.CM);
        dist = centerDistance ;
        if (dist.isNaN()) dist = 100.0;
        telemetry.addData("distance CM ", dist);
        telemetry.update();
        //creepCount++;

    }

}
//** moves latching pin right
public  void slideArmRight(double secs){
    liftPin.setPosition(0.3);

}
//** moves latching pin left
```

```
    public void slideArmLeft(double secs){
        liftPin.setPosition(0.88);

    }
    //** Moves left sampling arm to sampling position
    public void deployLeftSamplingArmToSample(){
        leftSampleArm.setPosition(0.88); //Position to prepare left arm to sample
    }
    //** Moves right sampling arm to sampling position
    public void deployRightSamplingArmToSample(){
        rightSampleArm.setPosition(0.05); //Position to prepare right  arm to sample
    }
    //** Moves left sampling arm to center PUSH position
    public void deployLeftSamplingArmToMiddle(){
        leftSampleArm.setPosition(0.02); //Position to prepare left arm to sample
        pause(1);//still needed-TEST??
    }
    //** Moves both sampling arms back to storage position
    public void storeBothSamplingArms(){
        storeRightSamplingArm();
        storeLeftSamplingArm();
    }
    //** Moves just Right sampling arm back to storage position
    public void storeRightSamplingArm(){
        rightSampleArm.setPosition(0.6); //Right Arm Stored position
    }
    //** Moves just Left sampling arm back to storage position
    public void storeLeftSamplingArm() {
        leftSampleArm.setPosition(0.2); //Left Arm Stored position
    }
    //** This will be improved as the plow itself is improved.
    public void dropPlow(){
        plow.setPower(.3);
        pause(1);
        plow.setPower(0);
    }
}
```