File: C:\Users\conhu\OneDrive\Documents\GitHub\MTE121_Final_Project\Main.c

```
//Sensor constants
const tSensors COLOR = S1;
const tSensors SONIC = S2;
const tSensors GYRO = S3;
const tSensors TOUCH = S4;

//Function prototypes
void calibrateSensors();
int getAbsoluteAngle();
int getAbsoluteAngle(int angle);
bool turnToAngle(int angle, int power, int & index);
bool turnToAngle(int angle, int power);
bool driveStraight(int time, int angle, int power, int & index);
void followLine(int power);
bool shoot(int angle, int power, int shootPower, int & index);
void victoryDance();


//RGB values and ranges
  //11-15
const int SHOOT_R = 15;
  //40-46
const int SHOOT_G = 50;
  //40-50
const int SHOOT_B = 55;

/* unused colors
  //10-13
const int TURN_R = 12;
  //41-49
const int TURN_G = 45;
*/
  //20-25
const int TURN_B = 30;

  //87-112
const int DRIVE_R = 110;
  //14-21
const int DRIVE_G = 18;
  //12-17
const int DRIVE_B = 18;

  //17-115
const int LINE_R = 30;
  //20-29
const int LINE_G = 30;
  //20-30
const int LINE_B = 30;

  //160-175
const TLegoColors FINISH = colorWhite;

  //100-130
const int BACKGROUND_R = 100;

/* unused colors
  //20-37
const int BACKGROUND_G = 28;
  //25-45
```

```
const int BACKGROUND_B = 40;
*/


//Movement constants
const int SHOOT_DIST = 60;
const int SONIC_TOL = 5;
const int SHOT_LG = 790;
const int ANGLE_TOL = 2;
const int FWD = 1;
const int BACK = -1;
const int TURN_SPEED_FACTOR = 2;

const int DATA_LG = 7;
const int DATA_ROW = 2;

//Sets all sensors to the correct ports and modes
void calibrateSensors()
{
  SensorType[COLOR]=sensorEV3_Color;
  wait1Msec(50);

  SensorMode[COLOR]=modeEV3Color_Color;

  wait1Msec(50);
  SensorType[SONIC]=sensorEV3_Ultrasonic;
  wait1Msec(50);
  SensorType[GYRO]=sensorEV3 Gyro;
  wait1Msec(50);
  SensorMode[GYRO]=modeEV3Gyro Calibration;
  wait1Msec(50);
  SensorMode[GYRO]=modeEV3Gyro RateAndAngle;
  wait1Msec(50);
  SensorType[TOUCH]=sensorEV3 Touch;
  wait1Msec(50);
}

//Takes in the gryo degrees and returns that angle in terms of 1 to 360 degrees
int getAbsoluteAngle()
{
  int angle = getGyroDegrees(GYRO);

  angle %= 360;

  if(angle < 0)
    angle += 360;

  return angle;
}

//Takes a given angle and returns that angle in terms of 1 to 360 degrees
int getAbsoluteAngle(int angle)
{
  angle %= 360;

  if(angle < 0)
    angle += 360;

  return angle;
```

```
}

//Given an angle, a motor power and the index
//it will turn the robot to that angle at the given power
//and then increment the index
bool turnToAngle(int angle, int power, int & index)
{
  angle = getAbsoluteAngle(angle);

  motor[motorA] = -power;
  motor[motorD] = power;
  bool fail = false;
  while (abs(getAbsoluteAngle() - angle) > ANGLE_TOL && !fail)
  {
    if (SensorValue[TOUCH] == 1)
    {
      fail = true;
    }
  }

  motor[motorA] = motor[motorD] = 0;
  if(!fail)
    index++;
  return !fail;
}

//Given an angle, a motor power and the index
//it will turn the robot to that angle at the given power
bool turnToAngle(int angle, int power)
{
  angle = getAbsoluteAngle(angle);
  bool fail = false;
  motor[motorA] = -power;
  motor[motorD] = power;
  while (abs(getAbsoluteAngle() - angle) > ANGLE_TOL && !fail)
  {
    if (SensorValue[TOUCH] == 1)
    {
      fail = true;
    }
  }

  motor[motorA] = motor[motorD] = 0;

  return !fail;
}

//Given a time angle, power and index the robot will turn to the
//given angle, drive for the given time in miliseconds at the given power
//and then increment the index
bool driveStraight(int time, int angle , int power, int & index)
{
  angle = getAbsoluteAngle(angle);
  bool fail = false;

  if(time < 0)
    fail = true;

  if(!fail)
```

```
  {
    turnToAngle(angle, power / TURN_SPEED_FACTOR);

    motor[motorA] = motor[motorD] = power;

    clearTimer(T1);

    while (time1[T1] < time && !fail)
    {
      if (SensorValue[TOUCH] == 1)
        fail = true;
    }

  motor[motorA] = motor[motorD] = 0;
  turnToAngle(angle, power / TURN_SPEED_FACTOR);
  if(!fail)
    index++;
  }
  return !fail;
}

//Given the motor power and direction
//the function will drive the shooter
//motor at the given power either
//forward or backwards
bool runShooter(int power, int dir)
{
  nMotorEncoder[motorB] = 0;
  motor[motorB] = -power * dir;
  bool fail = false;

  while(abs(nMotorEncoder[motorB]) < SHOT LG && !fail)
  {
    if (SensorValue[TOUCH] == 1)
      fail = true;
  }
  motor[motorB] = 0;
  return !fail;
}

//Given the target angle, power, shooter power and index
//the function will record its start position, turn to
//the given angle, move to the set distance, fire and then
//return to the track, and increment the index
bool shoot(int angle, int power, int shootPower, int & index)
{
  angle = getAbsoluteAngle(angle);
  bool fail = false;
  bool reverse = false;
  int startAng = getAbsoluteAngle();
  displayString(5, "%d", angle);

  turnToAngle(angle, power / TURN_SPEED_FACTOR);

  nMotorEncoder[motorA] = nMotorEncoder[motorD] = 0;

  playTone(400, 15);
  wait10Msec(15);
```

```
  if(SensorValue(SONIC) < SHOOT_DIST)
  {
    power *= -1;
    reverse = true;
  }

  motor[motorA] = motor[motorD] = power;
  while(abs(SensorValue(SONIC) - SHOOT_DIST) > SONIC_TOL && !fail)
  {
    displayString(4, "%d : %d : %d", SensorValue(SONIC), SHOOT_DIST,
    SensorValue(SONIC) - SHOOT_DIST);
    if (SensorValue[TOUCH] == 1)
      fail = true;

  }

  motor[motorA] = motor[motorD] = 0;

  int dist = nMotorEncoder[motorA];

  if(!runShooter(shootPower, FWD))
    return false;

  if(!runShooter(shootPower, BACK))
    return false;

  turnToAngle(getGyroDegrees(GYRO) + 180, power / TURN_SPEED_FACTOR);

  nMotorEncoder[motorA] = 0;
  if(!reverse)
  {
    motor[motorA] = motor[motorD] -power;
    while(nMotorEncoder[motorA] < dist && !fail)
    {
      if (SensorValue[TOUCH] == 1)
        fail = true;

    }
  } else
  {
    playTone(400, 15);
    wait10Msec(15);
    motor[motorA] = motor[motorD] = power;
    while(nMotorEncoder[motorA] > dist - 5 && !fail)
    {
      if (SensorValue[TOUCH] == 1)
        fail = true;

    }
  }

  turnToAngle(startAng, power / TURN_SPEED_FACTOR);
  if(!fail)
    index++;
  return !fail;
}

//Given the motor power, the function
//will decide whether to move left or right
```

```
//based on whether the line boundry is detected
void followLine(int power)
{

  int red = 0, blue = 0, green = 0;
  getColorRawRGB(COLOR, red, blue, green);
    if (red < BACKGROUND R && blue < LINE B)
    {
      motor[motorA]= 0;
      motor[motorD]= power;
    }
    else
    {
      motor[motorA] = power;
      motor[motorD] = 0;
    }

    wait1Msec(220);

    motor[motorA] = motor[motorD] = 0;
}

//The robot will play a repeating sound
//then generate a sound of increase pitch
//and spin untill the bumper is pressed
void victoryDance()
{
  bool fail = false;

  for(int x = 0; x < 10 && !fail; x++)
  {
    if(SensorValue(TOUCH) == 1)
        fail = true;

    playTone(250, 15);
    wait10Msec(16);

    playTone(500, 20);
    wait10Msec(21);
  }

  for(int x = 0; x <20000 && !fail; x += 4)
  {
    playTone(x, 1);
    motor[motorA] = -100;
    motor[motorD] = 100;
    wait10Msec(1);
    if(SensorValue[TOUCH] == 1)
      fail = true;
  }
  motor[motorA] = motor[motorD] = 0;
}

int const power = 20;

task main()
{
  calibrateSensors();
```

File: C:\Users\conhu\OneDrive\Documents\GitHub\MTE121_Final_Project\Main.c

```
  int moveData[DATA_ROW][DATA_LG] = {{270, 145, 325, 170, 270, 150, 45},
                                     {1,2,3,800,5000, 6, 7}};

  int index = 0;
  int shooterPower = 100;

  bool fail = false;
  while(!SensorValue(TOUCH) &&
        SensorValue(COLOR) != (int)FINISH && !fail && index < 10)
  {
    int red= 0 , blue = 0, green = 0;

    followLine(power);

    getColorRawRGB(COLOR,red,blue,green);
    displayString(3, "%d    %d    %d", red, green, blue);


    if(blue < SHOOT_B && red < SHOOT_R && green < SHOOT_G && blue > TURN_B)
    {
      if(index == 2)
        turnToAngle(135, power);

      motor[motorA] = motor[motorD] = 20;
      wait1Msec(750);
      motor[motorA] = motor[motorD] = 0;

      displayString(1, "SHOOT GREEN");
      if(!shoot(moveData[0][index], power, shooterPower, index))
        fail = true;


    } else

    if(red < LINE_R && blue < TURN_B && green > LINE_G)
    {

      displayString(1, "TURN BLUE");
      if(!turnToAngle(moveData[0][index], power / TURN_SPEED_FACTOR, index))
        fail = true;

      motor[motorA] = motor[motorD] = -20;
      wait1Msec(500);
      motor[motorA] = motor[motorD] = 0;

    } else

  if(blue < DRIVE_B && green < DRIVE_G && red < DRIVE_R && red > LINE_R)
  {
    motor[motorA] = motor[motorD] = 20;
    wait1Msec(500);
    motor[motorA] = motor[motorD] = 0;

    displayString(1, "DRIVE RED %d", index);
    if(!driveStraight(moveData[1][index], moveData[0][index], power, index))
      fail = true;

  }
```

```
    eraseDisplay();

  }

  victoryDance();
}
```