File: D:\Autonomous\autonomous-includes\autonomousTasks.h

```c
/////////////////////////////////////
//
//  AUTONOMOUS PROGRAM FUNCTIONS AND SUBROUTINES
//  CODE BY FTC TEAM# 5029
//  github.com/samohtj/PowerstackersFTC-5029
//  powerstackersftc.weebly.com
//  UPDATED 3-22-2014
//
////////////////////////////////////


#include "multiplexer.h"
#include "hitechnic-compass.h"
                                                                    // --GLOBAL VARIABLES
int lightThreshold = 470;                                     // Light threshold (stop after this value)
const int irThresh = 180;                                           // Infra-red threshold (stop after this value)
const int turnSpeed = 50;                                           // Speed of motors while turning
long startEncoderPos = 0;                                           // Encoder position at the start of the match

                                                                    // --SET ALL MOTORS TO INPUT VALUE
void allMotorsTo(int i){
  motor[mDriveLeft]   = i;
  motor[mDriveRight]  = i;
  motor[mBsAngle]     = i;
  motor[mBsConveyor]  = i;
  motor[mFlagRaise1]  = i;
  motor[mFlagRaise2]  = i;
}
                                                                    // --SET ALL DRIVE MOTORS TO INPUT VALUE
void driveMotorsTo(int i){
  motor[mDriveLeft]   = i;
  motor[mDriveRight]  = i;
}
                                                                    // --CONVERT INCHES TO ENCODER TICKS
long inchesToTicks(float inches){
  return (long) inches * (1350 / (4 * PI));
}
                                                                    // --CONVERT ENCODER TICKS TO INCHES
float ticksToInches(long ticks){
  return (float) ticks / (1350 / (4 * PI));
}
                                                                    // -- MOVE FORWARD A CERTAIN DISTANCE IN ENCODER TICKS
void goTicks(long ticks, int speed){
  long target = nMotorEncoder[mDriveRight] + ticks;                 // Calculate the target encoder value (current distance + dis
```

```
                                                                          // Print some relevant information to the debug stream
                                                                          // (Target encoder, current encoder, distance in inches, spee
  writeDebugStreamLine("-- MOVING TICKS --\ntarget: %5.2f, current:%5.2f (%d inches) (speed: %d)",
    target, nMotorEncoder[mDriveRight], ticksToInches(ticks), speed);
                                                                          // Create a modifier for the right wheel, since it spins fast
  float leftMotorRatio = (float) 80 / 100;                                // (The right wheel speed will be set lower to compensate for

  if(ticks > 0){                                                          // If the distance is positive:
    while(nMotorEncoder[mDriveRight] < target){                           // While the current value is lower than the target:
      motor[mDriveRight] = (int) speed * leftMotorRatio;                  // Move forwards
      motor[mDriveLeft] = speed;
    }
  }

  else{                                                                   // If the distance is negative:
    while(nMotorEncoder[mDriveRight] > target){                           // While the current value is higher than the target:
      motor[mDriveRight] = (float) -1 * (speed * leftMotorRatio);         // Move backwards
      motor[mDriveLeft] = -1 * speed;
    }
  }

  allMotorsTo(0);                                                         // Stop the motors
  writeDebugStreamLine("final:  %5.2f", nMotorEncoder[mDriveRight]);// Print the final encoder value
}
                                                                          // --TURN TO A SPECIFIC DEGREE ANGLE
void turnDegrees(float degreesToTurn, int turnStrength){
  // Entering 80 degrees will turn 90 degrees. Loss of about 10%
  float degreesSoFar = 0;                                                 // Degrees turned thus far
  int leftTurnStrength = turnStrength/* + 15*/;
  int initialTurnReading = HTGYROreadRot(sGyro);                          // Take an initial reading from the gyro
                                                                          // Print some info
  writeDebugStreamLine("-- TURNING --\ninitial reading: %d\nTarget angle: %2.2f",
    initialTurnReading,
    degreesToTurn);

  if (degreesToTurn > 0){                                                 // If the degree measure is positive:
    motor[mDriveLeft] = -1 * turnStrength;                                // Turn counterclockwise
    motor[mDriveRight] = turnStrength;
    writeDebugStreamLine("Decided to turn counterclockwise");
    }

  else{                                                                   // If the degree measure is negative:
    motor[mDriveLeft] = turnStrength;                                     // Turn clockwise
    motor[mDriveRight] = -1 * turnStrength;
```

```
    writeDebugStreamLine("Decided to turn clockwise");
  }

  while (abs(degreesSoFar) < abs(degreesToTurn)){                     // While the degrees we've turned is less than the target:
    wait10Msec(1);                                                    // Let some time pass
                                                                      // Edit the current gyro reading
    int currentGyroReading = HTGYROreadRot(sGyro) - initialTurnReading;
    nxtDisplayTextLine(7, "degreesSoFar: %d", degreesSoFar);
    degreesSoFar = degreesSoFar + (currentGyroReading * 0.01);        // Calculate the degrees turned so far (d=r*t)
    //writeDebugStreamLine("Currentangle: %d", degreesSoFar);           // Print the current degree measure to the debug stream
  }

  driveMotorsTo(0);                                                   // Stop the motors
  writeDebugStreamLine("final angle: %2.2f", degreesSoFar);           // Print the final degree measure
}
                                                                      // --CONTINUOUSLY RUN AND SHOW IMPORTANT INFORMATION
task showDebugInfo(){
  while(true){
    nxtDisplayTextLine(0, "mtrL:%d", motor[mDriveLeft]);  // Left motor encoder
    nxtDisplayTextLine(1, "mtrR:%d", motor[mDriveRight]);// Right motor encoder
    nxtDisplayTextLine(2, "LiL:%d", rawLightLeft);                    // Left light sensor
    nxtDisplayTextLine(3, "LiR:%d", rawLightRight);                   // Right light sensor
    nxtDisplayTextLine(5, "irRL:%d,%d",                               // IR seekers
      irStrengthLeft,
      irStrengthRight);
    nxtDisplayTextLine(6, "HighestIR:%d", irStrengthRight);           // Maximum IR signal
  }
}
                                                                      // --BRICK FLIPPER VARIABLES
const short blockDropLeftStart = 0;                                   // Starting (down) position
const short blockDropRightStart = 245;


const short blockDropLeftIdle = 128;                                  // Idle (out of the way) position
const short blockDropRightIdle = 128;

const short blockDropLeftDrop = 180;                                  // Dropping (extended) position
const short blockDropRightDrop = 32;

const short conveyorTightStart = 150;                                 // Conveyor activated and deactivated (start) position
//const short conveyorTightActive = 170;
                                                                      // --PUT FLIPPERS IN "DROP" POSITION
void blockDrop(){
  servo[rBlockDropLeft] = blockDropLeftDrop;
```

```
  servo[rBlockDropRight] = blockDropRightDrop;
}
                                                                    // --PUT FLIPPERS IN "RETRACTED" POSITION
void blockRetract(){
  servo[rBlockDropLeft] = blockDropLeftStart;
  servo[rBlockDropRight] = blockDropRightStart;
}
                                                                    // --PUT FLIPPERS IN "IDLE" POSITION
void blockIdle(){
  servo[rBlockDropLeft] = blockDropLeftIdle;
  servo[rBlockDropRight] = blockDropRightIdle;
}
                                                                    // --PUT THE BLOCK IN THE BASKET
void placeBlock(int basketPos){
  writeDebugStreamLine("-- PLACING BLOCK --");

  if(basketPos == 0 || basketPos == 1)                              // If we are at the first or second baskets:
    goTicks(inchesToTicks(8), 25);                                  // Adjust robot position
  if(basketPos == 2 || basketPos == 3)
    goTicks(inchesToTicks(6), 25);

  servo[rBlockDropLeft] = blockDropLeftDrop;                        // Extend the servos and drop the blocks
  servo[rBlockDropRight] = blockDropRightDrop;

  wait10Msec(100);                                                  // Give the servos time to extend

  servo[rBlockDropLeft] = blockDropLeftStart;                       // Put the servos into the rest position
  servo[rBlockDropRight] = blockDropRightStart;
}

                                                                      // --CALIBRATE THE LIGHT SENSORS FOR THE MATS
void calibrateLightSensors(){
  if(rawLightLeft == 0 || rawLightRight == 0){
    lightThreshold = 450;
    writeDebugStreamLine("No light signal detected. Threshold defaulted to %d", lightThreshold);
    return;
  }
  int matReading = (int) (rawLightLeft + rawLightRight) / 2;        // Average the right and left sensor values
  lightThreshold = matReading + 75;                                 // Add 75 to the average
  writeDebugStreamLine("Light level of mat: %d\nSet threshold to %d",// Print the mat light level and the threshold
  matReading, lightThreshold);
}

                                                                    // --FIND THE WHITE LINE
```

```
void findWhiteLine(){
  writeDebugStream("-- FINDING WHITE LINE --\n");
  bool foundLineLeft = false;                                      // Store whether the white line has been found
  bool foundLineRight = false;
  calibrateLightSensors();

  int maxLight = 0;                                                // Maximum signal detection:
  ClearTimer(T1);                                                  // Clear the timer
  motor[mDriveLeft] = 25;
  motor[mDriveRight] = 25;
  while(!foundLineLeft || !foundLineRight){                        // While neither line has been found:
    //if(time100[T1] % 10 == 0)                                    // Every 1 second:
    //  writeDebugStreamLine("maxLight == %d", maxLight);          // Print maximum detected value to the debug stream
    //if(rawLightLeft > maxLight)                                  // If current left light sensor value is greater than previou
    //  maxLight = rawLightLeft;                                   // Set new maximum
    //if(rawLightRight > maxLight)                                 // If current right light sensor value is greater than previo
    //  maxLight = rawLightRight;                                  // Set new maximum

    if(rawLightLeft > lightThreshold && !foundLineLeft){           // If the sensor value is above the threshold:
      foundLineLeft = true;                                        // Set the "found" flag to true
      writeDebugStreamLine("Found white line on the left. Detected value: %d Threshold value: %d",
      rawLightLeft, lightThreshold);                               // Print the detected value and the threshold value
      motor[mDriveLeft] = 0;                                       // Set the motor to 0
    }
    if(rawLightRight > lightThreshold && !foundLineRight){         // If the sensor value is above the threshold:
      foundLineRight = true;                                       // Set the "found" flag to true
      writeDebugStreamLine("Found white line on the right. Detected value: %d Threshold value: %d",
      rawLightRight, lightThreshold);                              // Prin the detected value and the threshold value
      motor[mDriveRight] = 0;                                      // Set the motor to 0
    }
  }
  motor[mDriveLeft] = 0;                                           // Set both motors to 0
  motor[mDriveRight] = 0;
}
                                                                   // --INITALIZE THE ROBOT
void initializeRobot(){
  allMotorsTo(0);                                                  // Set all the motors to 0
  calibrateLightSensors();
  nMotorEncoder[mDriveLeft] = 0;                                   // Set initial encoder vlaue to 0
  servo[rBlockDropLeft] = blockDropLeftStart;                      // Put the flipper servos in the start postion
  servo[rBlockDropRight] = blockDropRightStart;
  servo[rConveyorTight] = conveyorTightStart;                      // Put the conveyor tension servo in the start position
  writeDebugStreamLine("\n\n -- NEW INSTANCE -- \n\n");            // Print a section header to the debug stream
}
```

```
                                                                    // --FIND THE IR BEACON WITH INCREMENTAL MOVEMENT
void findIrIncremental(){
  if(irStrengthLeft > 600){
    writeDebugStreamLine("///////////////////\n//\n// MULTIPLEXER BATTERY DEAD\n//\n///////////////////");
    PlaySound(soundFastUpwardTones);
    goTicks(inchesToTicks(3), 25);
    placeBlock(0);
    return;
  }
// Store the distance from the starting position to each basket:
  long blockDistancesCumulative[3] = {startEncoderPos + inchesToTicks(20),
    startEncoderPos + inchesToTicks(43),
    startEncoderPos + inchesToTicks(53)};

  goTicks(inchesToTicks(10), 25);                                   // Move up to the first basket
  writeDebugStreamLine("At first basket, ready to start.\n");       // Print a "ready" message to the debug stream
  PlaySound(soundBeepBeep);                                         // Play a "ready" sound

  for(int i = 0; i <= 3; i++){                                      // Loop through four times:

    if(irStrengthLeft > irThresh || irStrengthRight > irThresh){    // If the signal to either IR seeker is above the threshold:
      placeBlock(i);                                                // Place the block in the basket (pass the bakset number)
      writeDebugStreamLine("Block placed in basket number %d. Detected value: %d Threshold value: %d",
      i+1, (irStrengthLeft > irStrengthRight)? irStrengthLeft : irStrengthRight, irThresh);           // Print a "finished" messa
      break;                                                        // Break out of the loop
    }

    else{                                                           // If both signals are below the threshold:
      goTicks(blockDistancesCumulative[i] -                         // Go to the next basket
      nMotorEncoder[mDriveRight], 25);
      writeDebugStreamLine("Going to next basket");
    }
                                                                    // Print the final basket number, the detected value, and the
    writeDebugStreamLine("\n-- BASKET #%d --\nCurrent Value: %d. Need %d to stop.",
    i+1, (irStrengthLeft > irStrengthRight)? irStrengthLeft:irStrengthRight, irThresh);
  }
}

int degreesToInches(int degrees){
  float radians = degreesToRadians(degrees);
  writeDebugStreamLine("Calculated %3.2f inches", radians * 8.75);
  return (int) radians * 9;
}
```

File: D:\Autonomous\autonomous-includes\autonomousTasks.h

```
void turnTicks(long ticks, int speed){
  long ticksTarget = nMotorEncoder[mDriveRight] + ticks;
  writeDebugStreamLine("Starting ticks: %d Ending ticks: %d Difference: %d", nMotorEncoder[mDriveRight],
  ticksTarget, ticksTarget - nMotorEncoder[mDriveRight]);
  if(ticks > 0){
    while(nMotorEncoder[mDriveRight] < ticksTarget){
      motor[mDriveLeft] = -1 * speed;
      motor[mDriveRight] = speed;
    }
  }else{
    while(nMotorEncoder[mDriveRight] > ticksTarget){
      motor[mDriveLeft] = speed;
      motor[mDriveRight] = -1 * speed;
    }
  }
  writeDebugStreamLine("Actual ending value: %d Error: %d", nMotorEncoder[mDriveRight],
  ticksTarget - nMotorEncoder[mDriveRight]);
  motor[mDriveLeft] = 0;
  motor[mDriveRight] = 0;
}
```