////////////////////////////////////////////////////////////////////////////////

// Radio functions

////////////////////////////////////////////////////////////////////////////////

void DashBot::clearRadioPacket(void)

{

for(byte i = 0; i< RADIO\_PACKET\_LENGTH; i++)

receivedRadioPacket[i] = 0;

}

boolean DashBot::readRadioPacket(void)

{

//if there's a full packet, read it and update the global packet variable.

//if there's not enoufgh data yet, do nothing and return a false

if(Serial1.available() >= RADIO\_PACKET\_LENGTH)

{

//digitalWrite(ledGreen, HIGH);

for (byte i = 0; i < RADIO\_PACKET\_LENGTH; i++)

{

receivedRadioPacket[i] = Serial1.read();

}

lastPacketTime = millis();

return true;

}

else

{

return false;

}

}

void DashBot::executeRadioCommand(void)

{

switch (receivedRadioPacket[0]) {

case ALL\_STOP:

mode = STOP\_MODE;

allStop();

clearRadioPacket();

auto\_flag = 0;

break;

case SET\_NAME:

setName();

break;

case JOYSTICK\_DRIVE:

// joystick drive

mode = JOYSTICK\_MODE;

directDrive(receivedRadioPacket[1], receivedRadioPacket[2], receivedRadioPacket[3], receivedRadioPacket[4]);

clearRadioPacket();

break;

case GYRO\_DRIVE:

// gyro-assisted driving

stabilizedDrive();

break;

case SET\_EYE\_COLOR:

setEyeColor(receivedRadioPacket[1], receivedRadioPacket[2], receivedRadioPacket[3]);

clearRadioPacket();

break;

case SEND\_INFO\_PACKET:

// send name and color to iOS device

sendInfoPacket();

break;

case SET\_INFO\_PACKET:

// set whether a name/color or sensor packet is being sent

setInfoPacketMode();

break;

case EXECUTE\_AUTO\_MODE:

executeAutoMode();

clearRadioPacket();

break;

default:

debugBlinkOn();

clearRadioPacket();

}

}

// executes one of Dash's autonomous behaviors

void DashBot::executeAutoMode(void){

auto\_flag = 1; // indicates an autonomous mode is on, if an all\_stop is called, resets to zero to end auto behavior

switch (receivedRadioPacket[1]) {

case DASH\_TEST:

dashTest();

Serial1.write('3');

case DASH\_CIRCLE:

dashCircle();

Serial1.write('3');

break;

case DASH\_FIG\_8:

dashFig8();

Serial1.write('3');

break;

case DASH\_DANCE:

dashDance();

Serial1.write('3');

break;

case DASH\_BUMP:

dashBump();

Serial1.write('3');

break;

default:

setEyeColor(0,0,0);

}

}

//sends sensor data to mobile device

void DashBot::sendInfoPacket(void){

infoPacketTime = millis();

Serial1.write(INFO\_PACKET); //2 means this is an info packet

Serial1.write(mode);

// send

int currentYaw = readGyroDeg();

if (currentYaw < 0)

currentYaw = -currentYaw;

Serial1.write(highByte(currentYaw));

Serial1.write(lowByte(currentYaw));

int currentAmbientLight = readAmbientLight();

Serial1.write(highByte(currentAmbientLight));

Serial1.write(lowByte(currentAmbientLight));

int currentProxLeft = readLeftIRsensor();

Serial1.write(highByte(currentProxLeft));

Serial1.write(lowByte(currentProxLeft));

int currentProxRight = readRightIRsensor();

Serial1.write(highByte(currentProxRight));

Serial1.write(lowByte(currentProxRight));

Serial1.write(motor\_right\_backward\_value);

Serial1.write(motor\_right\_forward\_value);

Serial1.write(motor\_left\_backward\_value);

Serial1.write(motor\_left\_forward\_value);

}

// send robot name and color to iOS device

void DashBot::sendNamePacket(void){

Serial1.write(NAME\_PACKET); //1 means this is a name packet

Serial1.write(robotType);

Serial1.write(robotColor);

Serial1.write(codeVersion);

for (byte i = 0; i < MAX\_NAME\_LENGTH; i++){

Serial1.write(robotName[i]);

}

}