```
#pragma config(Hubs,  S1, HTMotor,  HTMotor,  HTMotor,  HTMotor)
#pragma config(Hubs,  S2, HTServo,  none,     none,     none)
#pragma config(Sensor, S1,     ,               sensorI2CMuxController)
#pragma config(Sensor, S2,     ,               sensorI2CMuxController)
#pragma config(Sensor, S4,     HTSMUX,         sensorI2CCustom)
#pragma config(Motor,  motorA,          ,           tmotorNXT, openLoop)
#pragma config(Motor,  motorB,          ,           tmotorNXT, openLoop)
#pragma config(Motor,  motorC,          ,           tmotorNXT, openLoop)
#pragma config(Motor,  mtr_S1_C1_1,     IN,         tmotorTetrix, openLoop, reversed)
#pragma config(Motor,  mtr_S1_C1_2,     Lift,       tmotorTetrix, PIDControl, reversed, encoder)
#pragma config(Motor,  mtr_S1_C2_1,     IN2,        tmotorTetrix, openLoop, reversed)
#pragma config(Motor,  mtr_S1_C2_2,     motorZ,     tmotorTetrix, PIDControl, encoder)
#pragma config(Motor,  mtr_S1_C3_1,     BL,         tmotorTetrix, PIDControl, reversed, encoder)
#pragma config(Motor,  mtr_S1_C3_2,     BR,         tmotorTetrix, PIDControl, encoder)
#pragma config(Motor,  mtr_S1_C4_1,     FL,         tmotorTetrix, PIDControl, encoder)
#pragma config(Motor,  mtr_S1_C4_2,     FR,         tmotorTetrix, PIDControl, reversed, encoder)
#pragma config(Servo,  srvo_S2_C1_1,    IRVert,           tServoStandard)
#pragma config(Servo,  srvo_S2_C1_2,    IRHoriz,          tServoStandard)
#pragma config(Servo,  srvo_S2_C1_3,    YSplitClamp,      tServoStandard)
#pragma config(Servo,  srvo_S2_C1_4,    US,               tServoStandard)
#pragma config(Servo,  srvo_S2_C1_5,    HoodExtend,       tServoContinuousRotation)
#pragma config(Servo,  srvo_S2_C1_6,    HoodRotate,       tServoStandard)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

#include "JoystickDriver.c"  //Include file to "handle" the Bluetooth messages.
#include "hitechnic-sensormux.h"
#include "drivers/hitechnic-gyro.h"
#include "hitechnic-irseeker-v2.h" //Driver for IRSeeker
#include "lego-ultrasound.h"
//#include "HTIRS-driver.h"
#include "common-mmux.h"
//#include "common.h"
#include "drivers/lego-light.h"
#include "drivers/hitechnic-colour-v2.h"

//#define HTIRS2_DSP_MODE = 1200;

const tMUXSensor LEGOUS = msensor_S4_1;
const tMUXSensor HTGYRO = msensor_S4_2;
const tMUXSensor HTIRS2 = msensor_S4_3;
const tMUXSensor light = msensor_S4_4;
//const float FUDGE_FACTOR = 1;
const float TICK_RATIO = 89.75;
const float SERVO_RATIO = 1.4167;
//const float WHEEL_RATIO = 1;
//const float CONVERT_LENGTH_RATIO = 1;
//const float GEAR_RATIO = 1;
```

```
//--------[ NOTE: #PRAGMAS MUST COME BEFORE ANY COMMENTS OR STATEMENTS! ]--------

//---------------------[ PROGRAM CREDITS ]-----------------------------------------------
//        TEAM                                                    : "eTREC Robotics" Team #6977
//        ROBOT NAME                              : Zen-42 (Previously #6914-B)
//        PLATFORM TYPE                           :       Lego Mindstorms NXT / Pitsco TETRIX
//        FILE NAME                                       : Zen-42 eTrec - Auto.c
//        FILE CREATED ON                 : 11.14.2014 - 9:43PM
//        AUTHOR(S)                                       : Nikolaus Prusinski
//        DESCRIPTION                             : This is a model autonomous program that will be used on Zen-42 and
//        DESCRIPTION (CONT):   later transferred to eTREC's Robot for the final competition.
//        REVISION HISTORY        : See revision history at the end of the file
//---------------------------------------------------------------------------------------

//---------------------[ INITIALIZE GLOBAL VARIABLES ]-----------------------------------

int Ldist = 0;
int LdistCompareVal = 0;
int RampLdist = 0;
int RampLdistCompareVal = 0;
int Cdist = 0;
int CdistCompareVal = 0;
int RampCdist = 0;
int RampCdistCompareVal = 0;
int Rdist = 0;
int RdistCompareVal = 0;
int FloorCdist = 0;
int FloorCdistCompareVal = 0;
int _strDC = 0;
int _dirAC = 0;
int raw = 217;
int raw1 = 0;
int raw2 = 0;
int rawFloor = 0;
int dcS1, dcS2, dcS3, dcS4, dcS5 = 0;
int ColorSensorRed = 0;
int ColorSensorGreen = 0;
int ColorSensorBlue = 0;
int avgColorValue = 0;
int IRtestResult = 0;
//int servoDegrees = 0;
//int waitTime = 0;
int locationNumber = 0;
int generalDefenseNumber = 0;
int rampDefenseNumber = 0;
int encoderDefenseLeft = 0;
```

```
int encoderDefenseRight = 0;
int encoderDefenseForward = 0;
int encoderDefenseRampForward = 0;
int encoderDefenseRampLeft = 0;


//--------------------[ ROBOT INITIALIZATION ]-----------------------------------------

void initializeRobot()
{
	HTGYROstartCal(HTGYRO);
	ClearTimer(T1);
	ClearTimer(T2);
	ClearTimer(T3);
	ClearTimer(T4);
 return;
}

//--------------------[ FUNCTION 'Stop' ]-----------------------------------------------

void Stop()
	{
		motor[FL] = 0;
		motor[FR] = 0;
		motor[BL] = 0;
		motor[BR] = 0;
	}

//--------------------[ FUNCTION 'EncoderMove' ]----------------------------------------------

void EncoderMove(int TravelDist, int motorSpeed)
	{
		nMotorEncoder[FL] = 0;
		nMotorEncoder[FR] = 0;
		nMotorEncoder[BL] = 0;
		nMotorEncoder[BR] = 0;

		long numticks = 0;
		if(TravelDist > 0)
			{
				numticks =
(TravelDist*TICK_RATIO);//*GEAR_RATIO*FUDGE_FACTOR*WHEEL_RATIO*CONVERT_LENGTH_RATIO;
		writeDebugStreamLine(" TravelDist: %d", TravelDist);
		writeDebugStreamLine(" numticks: %d", numticks);
					writeDebugStreamLine(" nMotorEncoderTarget[FL]: %d",
nMotorEncoderTarget[FL]);
				while(nMotorEncoder[FL] < numticks)
```

```
                                {
                                        motor[FL] = motorSpeed;
                                        motor[FR] = motorSpeed;
                                        motor[BL] = motorSpeed;
                                        motor[BR] = motorSpeed;
                                }
                                writeDebugStreamLine("nMotorEncoder[FL]: %d", nMotorEncoder[FL]);

                                Stop();
                        }
                else
                {
                numticks =
(TravelDist*TICK_RATIO);//*GEAR_RATIO*FUDGE_FACTOR*WHEEL_RATIO*CONVERT_LENGTH_RATIO;
                writeDebugStreamLine(" TravelDist: %d", TravelDist);
                writeDebugStreamLine(" numticks: %d", numticks);

                writeDebugStreamLine(" nMotorEncoderTarget[FL]: %d", nMotorEncoderTarget[FL]);


                        while(nMotorEncoder[FL] > numticks)
                                {
                                        motor[FL] = -motorSpeed;
                                        motor[FR] = -motorSpeed*1.1;
                                        motor[BL] = -motorSpeed;
                                        motor[BR] = -motorSpeed*1.1;
                                }


                                writeDebugStreamLine("nMotorEncoder[FL]: %d", nMotorEncoder[FL]);

                                Stop();
                }
}
//---------------------[ DEFENSE SUBROUTINES BEGIN ]----------------------------------------------




//---------------------[ FUNCTION 'RampRobotForward' ]----------------------------------------------
void RampRobotForward()
{
        Stop();
        writeDebugStreamLine("Ramp Robot Forward");
        writeDebugStreamLine("RampCdist (1): %d", RampCdist);
        RampCdist = USreadDist(LEGOUS);
```

```
	wait1Msec(1000);
	RampCdistCompareVal = USreadDist(LEGOUS);
	if(RampCdist > RampCdistCompareVal)
	{
		nMotorEncoder[FL] = 0;
		nMotorEncoder[FR] = 0;
		nMotorEncoder[BL] = 0;
		nMotorEncoder[BR] = 0;
		while(RampCdist < 50)
		{
			motor[FL] = -35;
			motor[FR] = -35;
			motor[BL] = -35;
			motor[BR] = -35;
		}
		Stop();
		nMotorEncoder[FR] = encoderDefenseRampForward;
	}
	if(RampCdist == RampCdistCompareVal)
	{
		while(RampCdist < 50)
		{
			RampCdist = USreadDist(LEGOUS);
			writeDebugStreamLine("RampCdist: %d", RampCdist);
			motor[FL] = -20;
			motor[FR] = -20;
			motor[BL] = -20;
			motor[BR] = -20;
		}
		Stop();
	}
	else
	{
		wait1Msec(2000);
	}
}
//---------------------[ FUNCTION 'RampRobotLeft' ]-----------------------------------------------
void RampRobotLeft()
{
	Stop();
	writeDebugStreamLine("Ramp Robot Left");
	writeDebugStreamLine("RampLdist (1): %d", RampLdist);
	RampLdist = USreadDist(LEGOUS);
	wait1Msec(1000);
	ClearTimer(T2);
	while(time1[T2] < 5000)
	{
```

```
		motor[FL] = 0;
		motor[FR] = 0;
		motor[BL] = 0;
		motor[BR] = 0;
	}
	RampLdistCompareVal = USreadDist(LEGOUS);
	if(RampLdistCompareVal <= RampLdist)
	{
		//Do we Repeat?
		nMotorEncoder[FL] = 0;
		nMotorEncoder[FR] = 0;
		nMotorEncoder[BL] = 0;
		nMotorEncoder[BR] = 0;


		nMotorEncoder[FL] = encoderDefenseRampLeft;
	}
	if(RampCdist == RampCdistCompareVal)
	{
		//while(RampCdist < 50)
		//{
		//	RampCdist = USreadDist(LEGOUS);
		//	writeDebugStreamLine("RampCdist: %d", RampCdist);
		//	motor[FL] = -20;
		//	motor[FR] = -20;
		//	motor[BL] = -20;
		//	motor[BR] = -20;
		//}
}
		Stop();
}
//--------------------[ FUNCTION 'RobotLeft' ]-------------------------------------------------
void RobotLeft()
{
	Stop();
	writeDebugStreamLine("Robot Left");
	writeDebugStreamLine("Ldist (1): %d", Ldist);
	Ldist = USreadDist(LEGOUS);
	wait1Msec(1000);
	LdistCompareVal = USreadDist(LEGOUS);
	writeDebugStreamLine("LdistCompareVal: %d", LdistCompareVal);
	if(Ldist || LdistCompareVal < 0)
	{
		Ldist = USreadDist(LEGOUS);
	}
	if(Ldist < LdistCompareVal)
	{
```

```
		//Coming Toward US!!! Ahhhh!!!
		nMotorEncoder[FL] = 0;
		nMotorEncoder[FR] = 0;
		nMotorEncoder[BL] = 0;
		nMotorEncoder[BR] = 0;

		while(Ldist < 50)
		{
			Ldist = USreadDist(LEGOUS);
			motor[FL] = -20;
			motor[FR] = -20;
			motor[BL] = -20;
			motor[BR] = -20;
		}
		Stop();
		nMotorEncoder[FR] = encoderDefenseLeft;
	}
		if(Ldist == LdistCompareVal && Ldist || LdistCompareVal != 255)
	{
		//Coming Toward US!!! Ahhhh!!!
		nMotorEncoder[FL] = 0;
		nMotorEncoder[FR] = 0;
		nMotorEncoder[BL] = 0;
		nMotorEncoder[BR] = 0;

		while(Ldist < 50)
		{
			Ldist = USreadDist(LEGOUS);
			motor[FL] = -20;
			motor[FR] = -20;
			motor[BL] = -20;
			motor[BR] = -20;
		}
		Stop();
		nMotorEncoder[FR] = encoderDefenseLeft;
	}
	else
	{
		//Going Away. Good Bye! So long and thanks for all the fish!!
		wait1Msec(5000);
		//nMotorEncoder[FR] = encoderDefenseLeft;
	}

}
//---------------------[ FUNCTION 'RobotForward' ]-----------------------------------------------
void RobotForward()
{
```

```
Stop();
writeDebugStreamLine("Robot Forward");
writeDebugStreamLine("Cdist (1): %d", Cdist);
Cdist = USreadDist(LEGOUS);
wait1Msec(1000);
CdistCompareVal = USreadDist(LEGOUS);
writeDebugStreamLine("CdistCompareVal: %d", CdistCompareVal);
if(Cdist < CdistCompareVal)
{
	//Coming Toward US!!! Ahhhh!!!
	nMotorEncoder[FL] = 0;
	nMotorEncoder[FR] = 0;
	nMotorEncoder[BL] = 0;
	nMotorEncoder[BR] = 0;

	while(Cdist < 50)
	{
		Cdist = USreadDist(LEGOUS);
		motor[FL] = -20;
		motor[FR] = -20;
		motor[BL] = -20;
		motor[BR] = -20;
	}
	Stop();
	nMotorEncoder[FR] = encoderDefenseForward;
}
if(Cdist == CdistCompareVal && Cdist || CdistCompareVal != 255)
{
	//Coming Toward US!!! Ahhhh!!!
	nMotorEncoder[FL] = 0;
	nMotorEncoder[FR] = 0;
	nMotorEncoder[BL] = 0;
	nMotorEncoder[BR] = 0;

	while(Cdist < 50)
	{
		Cdist = USreadDist(LEGOUS);
		motor[FL] = -20;
		motor[FR] = -20;
		motor[BL] = -20;
		motor[BR] = -20;
	}
	Stop();
	nMotorEncoder[FR] = encoderDefenseForward;
}
if(Cdist || CdistCompareVal < 0)
{
```

```
		Cdist = USreadDist(LEGOUS);
	}
	else
	{
		//Going Away. Good Bye! So long and thanks for all the fish!!
		wait1Msec(5000);
		//nMotorEncoder[FR] = encoderDefenseForward;
	}
}
//---------------------[ FUNCTION 'RobotRight' ]-----------------------------------------------
void RobotRight()
{
	Stop();
	writeDebugStreamLine("Robot Right");
	writeDebugStreamLine("Rdist (1): %d", Rdist);
	wait1Msec(1000);
	Rdist = USreadDist(LEGOUS);
	wait1Msec(1000);
	RdistCompareVal = USreadDist(LEGOUS);
	writeDebugStreamLine("RdistCompareVal: %d", RdistCompareVal);
	if(Rdist < RdistCompareVal)
	{
		//Coming Toward US!!! Ahhhh!!!
		nMotorEncoder[FL] = 0;
		nMotorEncoder[FR] = 0;
		nMotorEncoder[BL] = 0;
		nMotorEncoder[BR] = 0;

		while(Rdist < 50)
		{
			Rdist = USreadDist(LEGOUS);
			motor[FL] = -20;
			motor[FR] = -20;
			motor[BL] = -20;
			motor[BR] = -20;
		}
		Stop();
		nMotorEncoder[FR] = encoderDefenseRight;
	}
		if(Rdist == RdistCompareVal && Rdist || RdistCompareVal != 255)
	{
		//Coming Toward US!!! Ahhhh!!!
		nMotorEncoder[FL] = 0;
		nMotorEncoder[FR] = 0;
		nMotorEncoder[BL] = 0;
		nMotorEncoder[BR] = 0;
```

```
		while(Rdist < 50)
		{
			Rdist = USreadDist(LEGOUS);
			motor[FL] = -20;
			motor[FR] = -20;
			motor[BL] = -20;
			motor[BR] = -20;
		}
		Stop();
		nMotorEncoder[FR] = encoderDefenseRight;
	}
	if(Rdist || RdistCompareVal < 0)
	{
		Rdist = USreadDist(LEGOUS);
	}
	else
	{
		//Going Away. Good Bye! So long and thanks for all the fish!!
		wait1Msec(5000);
		//nMotorEncoder[FR] = encoderDefenseReft;
	}
}

//---------------------[ FUNCTION 'FloorRobotForward' ]-----------------------------------------------
void FloorRobotForward()
{
	Stop();
	writeDebugStreamLine("Floor Robot Center");
	writeDebugStreamLine("FloorCdist (1): %d", FloorCdist);
	FloorCdist = USreadDist(LEGOUS);
	wait1Msec(500);
	ClearTimer(T1);
	while(time1[T1] < 5000)
	{
		motor[FL] = 0;
		motor[FR] = 0;
		motor[BL] = 0;
		motor[BR] = 0;
	}
	FloorCdistCompareVal = USreadDist(LEGOUS);
	if(FloorCdistCompareVal <= FloorCdist)
	{
		//Go Around?
		nMotorEncoder[FL] = 0;
		nMotorEncoder[FR] = 0;
		nMotorEncoder[BL] = 0;
		nMotorEncoder[BR] = 0;
```

```
	}
}
//---------------------[ FUNCTION 'Error' ]-----------------------------------------------
void Error(int lineNum, string errorTxt,)
{
		Stop();
		writeDebugStreamLine("Error");
		nxtDisplayTextLine(lineNum, "errorTxt");
		nxtDisplayTextLine(5, "_dirAC < 0 OR > 9");
		PlaySound(soundBeepBeep);
		LSsetActive(light);
		raw = LSvalRaw(light);
		writeDebugStreamLine(" raw: %d", raw);
		wait1Msec(5000);
}
//---------------------[ FUNCTIONS 'DEFENSE FUNCTIONS' ]----------------------------------------


void GeneralDistanceDecision()
	{
		switch(generalDefenseNumber)
		{
			case 1: RobotLeft(); break;
			case 2: RobotForward(); break;
			case 3: RobotRight(); break;
		}
}

void FindRobotLeft()
{
	servo[US] = 90;
	Ldist = USreadDist(LEGOUS);
	if(Ldist < 0)
	{
		string errorTxt = "US Fail Ldist < 0";
		Error(4, errorTxt);
	}
	if(Ldist < 50)
	{
		generalDefenseNumber = 1;
		GeneralDistanceDecision();
	}
}
void FindRobotCenter()
{
	servo[US] = 225;
	Cdist = USreadDist(LEGOUS);
```

```
	if(Cdist < 0)
	{
		string errorTxt = "US Fail Ldist < 0";
		Error(4, errorTxt);
	}
	if(Cdist < 50)
	{
		generalDefenseNumber = 2;
		GeneralDistanceDecision();
	}
}
void FindRobotRight()
{
	servo[US] = 255;
	Rdist = USreadDist(LEGOUS);
	if(Rdist < 0)
	{
		string errorTxt = "US Fail Ldist < 0";
		Error(4, errorTxt);
	}
	if(Rdist < 50)
	{
		generalDefenseNumber = 3;
		GeneralDistanceDecision();
	}
}



void RampDistanceDecision()
{
	switch(rampDefenseNumber)
	{
		case 1: RampRobotLeft(); break;
		case 2: RampRobotForward(); break;
	}
}


void RampFindRobotLeft()
{
	servo[US] = 90;
	RampLdist = USreadDist(LEGOUS);
	if(RampLdist < 0)
	{
		string errorTxt = "US Fail Ldist < 0";
		Error(4, errorTxt);
```

```
	}
	if(RampLdist < 50)
	{
		rampDefenseNumber = 1;
		RampDistanceDecision();
	}
}
void RampFindRobotCenter()
{
	writeDebugStreamLine("RampFindRobotCenter");
	servo[US] = 225;
	wait1Msec(1000);
	RampCdist = USreadDist(LEGOUS);
	wait1Msec(1000);
	writeDebugStreamLine("RampCdist: %d", RampCdist);
	//if(RampCdist < 0)
	//{
	//	string errorTxt = "US Fail Ldist < 0";
	//	Error(4, errorTxt);
	//}
	if(RampCdist < 50)
	{
		writeDebugStreamLine("RampCdist < 50");
		rampDefenseNumber = 2;
		RampDistanceDecision();
	}
	if(RampCdist == 255)
	{
		wait1Msec(1000);
		RampCdist = USreadDist(LEGOUS);
		wait1Msec(1000);
	}
}


void FloorFindRobotCenter()
{
		servo[US] = 225;
	FloorCdist = USreadDist(LEGOUS);
	if(FloorCdist < 0)
	{
		string errorTxt = "US Fail Ldist < 0";
		Error(4, errorTxt);
	}
	if(FloorCdist < 50)
	{
		FloorRobotForward();
```

```
	}
}
//---------------------[ DEFENSE FUNCTIONS/SUBROUTINES END ]-----------------------------------------------




//---------------------[ FUNCTION 'IRServoTurn' ]-----------------------------------------------

void IRServoTurn(int servoDegrees)
	{
		servo[TopHoriz] = (servoDegrees*SERVO_RATIO);
	}


//---------------------[ FUNCTION 'GyroTurn']----------------------
void GyroTurn(int degreesToTurn)
{
//HTGYROstartCal(HTGYRO);
float degreesSoFar = 0;
int initialTurnReading = HTGYROreadRot(HTGYRO);

//nSyncedMotors = synchBC;
//nSyncedTurnRatio = -100;
motor[FL] = -25*sgn(degreesToTurn);
motor[BL] = -25*sgn(degreesToTurn);
motor[FR] = 25*sgn(degreesToTurn);
motor[BR] = 25*sgn(degreesToTurn);

while(abs(degreesSoFar) < abs(degreesToTurn))
{
	wait1Msec(10);
	int currentHTGYROReading = HTGYROreadRot(HTGYRO) - initialTurnReading;
	degreesSoFar = degreesSoFar + currentHTGYROReading*.018;
	writeDebugStreamLine(" degreesSoFar: %d", degreesSoFar);
}
Stop();
}




//---------------------[ FUNCTION 'Function1' ]-----------------------------------------------

void Function1()
{
```

```
	FindRobotLeft();
		IRServoTurn(90);
		EncoderMove(-10,30);
		Stop();
		wait1Msec(500);
		GyroTurn(-40);
		Stop();
		wait1Msec(500);
		EncoderMove(17,30);
		wait1Msec(500);
		GyroTurn(50);
		wait1Msec(500);
		//EncoderMove(5,25);
nMotorEncoder[FL] = 0;
nMotorEncoder[FR] = 0;
nMotorEncoder[BL] = 0;
nMotorEncoder[BR] = 0;

while(nMotorEncoder[FL] < 2244)
{
	motor[FL] = 100;
	motor[FR] = 96;
	motor[BL] = 100;
	motor[BR] = 96;
}
	(HTIRS2readAllDCStrength(HTIRS2, dcS1, dcS2, dcS3, dcS4, dcS5));
	_strDC = dcS5;

	motor[FL] = 100;
	motor[FR] = 96;
	motor[BL] = 100;
	motor[BR] = 96;

	while(_strDC < 50)
	{
		(HTIRS2readAllACStrength(HTIRS2, dcS1, dcS2, dcS3, dcS4, dcS5));
		_strDC = dcS5;
		writeDebugStreamLine("_strDC: %d", _strDC);
	}

	Stop();
	EncoderMove(10,30);
	GyroTurn(105);
	wait1Msec(500);
	FindRobotCenter();
	FindRobotRight();
	servo[TopHoriz] = 90;
```

```
_dirAC = HTIRS2readACDir(HTIRS2);
while(_dirAC > 0)
{
	_dirAC = HTIRS2readACDir(HTIRS2);
	motor[FL] = 30;
	motor[FR] = 30;
	motor[BL] = 30;
	motor[BR] = 30;
}
	EncoderMove(-5,30);

	GyroTurn(105);
	wait1Msec(500);
	EncoderMove(7,15);

}

//----------------------[ FUNCTION 'Function3' ]-----------------------------------------------
void Function3()
{
	FindRobotLeft();
IRServoTurn(90);
EncoderMove(-4,15);
GyroTurn(-85);

	nMotorEncoder[FL] = 0;
nMotorEncoder[FR] = 0;
nMotorEncoder[BL] = 0;
nMotorEncoder[BR] = 0;

while(nMotorEncoder[FL] < 800)
{
	motor[FL] = 100;
	motor[FR] = 100;
	motor[BL] = 100;
	motor[BR] = 100;
}
	(HTIRS2readAllDCStrength(HTIRS2, dcS1, dcS2, dcS3, dcS4, dcS5));
	_strDC = dcS5;

	motor[FL] = 100;
	motor[FR] = 100;
	motor[BL] = 100;
	motor[BR] = 100;

	while(_strDC < 20)
	{
```

```
                (HTIRS2readAllACStrength(HTIRS2, dcS1, dcS2, dcS3, dcS4, dcS5));
                _strDC = dcS5;
                writeDebugStreamLine("_strDC: %d", _strDC);
        }

        Stop();
        EncoderMove(20,35);
GyroTurn(90);
FindRobotRight();
        wait1Msec(500);
        servo[TopHoriz] = 90;
_dirAC = HTIRS2readACDir(HTIRS2);
while(_dirAC > 0)
{
        _dirAC = HTIRS2readACDir(HTIRS2);
        motor[FL] = 30;
        motor[FR] = 30;
        motor[BL] = 30;
        motor[BR] = 30;
}
        EncoderMove(-5,30);

        GyroTurn(105);
        wait1Msec(500);
        EncoderMove(9,30);
        Stop();
        wait1Msec(2000);

        EncoderMove(-9,15);
        GyroTurn(35);
        EncoderMove(15,30);
        GyroTurn(-30);
        EncoderMove(35,100);
        Stop();
}

//---------------------[ FUNCTION 'Function2' ]-----------------------------------------------
void Function2()
{
        FindRobotLeft();
        IRServoTurn(90);
        EncoderMove(-4,15);
        GyroTurn(-8);

nMotorEncoder[FL] = 0;
nMotorEncoder[FR] = 0;
nMotorEncoder[BL] = 0;
```

```
nMotorEncoder[BR] = 0;

while(nMotorEncoder[FL] < 2244)
{
	motor[FL] = 100;
	motor[FR] = 100;
	motor[BL] = 100;
	motor[BR] = 100;
}
	(HTIRS2readAllDCStrength(HTIRS2, dcS1, dcS2, dcS3, dcS4, dcS5));
	_strDC = dcS5;

	motor[FL] = 100;
	motor[FR] = 100;
	motor[BL] = 100;
	motor[BR] = 100;

	while(_strDC < 50)
	{
		(HTIRS2readAllACStrength(HTIRS2, dcS1, dcS2, dcS3, dcS4, dcS5));
		_strDC = dcS5;
		writeDebugStreamLine("_strDC: %d", _strDC);
	}

	Stop();

	GyroTurn(90);
	FindRobotRight();
	servo[TopHoriz] = 90;
_dirAC = HTIRS2readACDir(HTIRS2);
while(_dirAC > 0)
{
	_dirAC = HTIRS2readACDir(HTIRS2);
	motor[FL] = 30;
	motor[FR] = 30;
	motor[BL] = 30;
	motor[BR] = 30;
}
	EncoderMove(-5,30);

	GyroTurn(105);
	wait1Msec(500);
	EncoderMove(7,30);
	Stop();


}
```

```
//----------------------[ FUNCTION 'Collect Sensor Data' ]-----------------------------------------------

void CollectSensorData()
{
		IRServoTurn(110);
		Cdist = USreadDist(LEGOUS);
		wait1Msec(100);
		writeDebugStreamLine(" Cdist: %d", Cdist);
		wait1Msec(500);
		servo[US] = 240;
		Rdist = USreadDist(LEGOUS);
		wait1Msec(100);
		if(Rdist == 255)
			{
				Rdist = 14;
			}
		_dirAC = HTIRS2readACDir(HTIRS2);
	 (HTIRS2readAllACStrength(HTIRS2, dcS1, dcS2, dcS3, dcS4, dcS5));
	 _strDC = (dcS1 + dcS2 + dcS3 + dcS4 + dcS5)/5;
		writeDebugStreamLine(" _dirAC: %d", _dirAC);
		writeDebugStreamLine(" _strDC: %d", _strDC);

		writeDebugStreamLine(" Rdist: %d", Rdist);

		writeDebugStreamLine("ColorSensorGreen: %d",ColorSensorGreen);
			writeDebugStreamLine("ColorSensorBlue: %d", ColorSensorBlue);
			writeDebugStreamLine("ColorSensorRed: %d", ColorSensorRed);


			//(HTCS2readRGB(HTCS2, ColorSensorRed, ColorSensorGreen, ColorSensorBlue));

				writeDebugStreamLine("ColorSensorGreen: %d",ColorSensorGreen);
			writeDebugStreamLine("ColorSensorBlue: %d", ColorSensorBlue);
				writeDebugStreamLine("ColorSensorRed: %d", ColorSensorRed);

		avgColorValue = (ColorSensorRed + ColorSensorGreen + ColorSensorBlue)/3;
		writeDebugStreamLine("avgColorValue: %d", avgColorValue);

}

	//----------------------[ FUNCTION 'WaitTime' ]-----------------------------------------------

void WaitTime(int waitTime)
	{
		wait1Msec(waitTime);
	}
```

```
//---------------------[ FUNCTION 'Decision' ]-----------------------------------------------
void Decision()
{
	if(_dirAC < 4 && _dirAC > 0)
	{
		writeDebugStreamLine("Position 1");
		IRtestResult = 1;
	}
	if(_dirAC == 4 || _dirAC == 5 || _dirAC == 6 || _dirAC == 8)
	{
		writeDebugStreamLine("Position 2");
		IRtestResult = 2;
	}
	if(_dirAC == 0)
	{
		writeDebugStreamLine("Position 3");
		IRtestResult = 3;
	}
	if(_dirAC < 0 || _dirAC > 9)
	{
	string errorTxt = "IR Fail";
	string errorTxtL2 = "Dir < 0 OR Dir > 9";
	Error(4, errorTxt);
	Error(5, errorTxtL2);
	wait1Msec(5000);
	}
	switch(IRtestResult)
	{
		case 1: Function1(); break;
		case 2: Function2(); break;
		case 3: Function3(); break;
	}
}




//---------------------[ FUNCTION 'BlueRamp' ]-----------------------------------------------

void BlueRamp()
	{
		writeDebugStreamLine("BlueRamp");
		//LSsetActive(light);
		//raw = LSvalRaw(light);
		//writeDebugStreamLine(" raw: %d", raw);
```

```
	motor[FL] = 30;
		motor[FR] = 30;
		motor[BL] = 30;
		motor[BR] = 30;
	while(raw < 235)
	{
		raw = LSvalRaw(light);
		wait1Msec(5);
		writeDebugStreamLine(" raw: %d", raw);
 }
	writeDebugStream("Floor");
	Stop();
	RampFindRobotLeft();
	RampFindRobotCenter();
	wait1Msec(100);

	motor[FL] = 25;
		motor[FR] = 25;
		motor[BL] = 25;
		motor[BR] = 25;
		while(raw > 220)
	{
		raw = LSvalRaw(light);
		wait1Msec(5);
 }
	writeDebugStream("See Line");
Stop();
wait1Msec(500);

nMotorEncoder[FL] = 0;
nMotorEncoder[FR] = 0;
nMotorEncoder[BL] = 0;
nMotorEncoder[BR] = 0;
while(nMotorEncoder[FL] < 700)
{
	motor[FL] = 25;
	motor[FR] = 25;
	motor[BL] = 25;
	motor[BR] = 25;
}
Stop();
servo[US] = 225;

WaitTime(2000);
GyroTurn(-90);
Stop();
FindRobotLeft();
```

```
FindRobotCenter();
FindRobotRight();
WaitTime(2000);
EncoderMove(17,30);
Stop();
FindRobotLeft();
FindRobotRight();
EncoderMove(17,30);
Stop();
wait1Msec(500);
//FindRobotLeft();
CollectSensorData();
Decision();

}



//---------------------[ FUNCTION 'RedRamp' ]-----------------------------------------

void RedRamp()
{
	writeDebugStreamLine("RedRamp");
		LSsetActive(light);
		raw = LSvalRaw(light);
		writeDebugStreamLine(" raw: %d", raw);

	motor[FL] = 30;
		motor[FR] = 30;
		motor[BL] = 30;
		motor[BR] = 30;
	while(raw > 232)
	{
		raw = LSvalRaw(light);
		wait1Msec(5);
		writeDebugStreamLine(" raw: %d", raw);
 }
	writeDebugStream("Floor");
	Stop();
	RampFindRobotLeft();
	RampFindRobotCenter();
	wait1Msec(100);

	motor[FL] = 25;
		motor[FR] = 25;
		motor[BL] = 25;
		motor[BR] = 25;
```

```
		while(raw < 300)
	{
		raw = LSvalRaw(light);
		wait1Msec(5);
 }
	writeDebugStream("See Line");
Stop();
wait1Msec(500);

nMotorEncoder[FL] = 0;
nMotorEncoder[FR] = 0;
nMotorEncoder[BL] = 0;
nMotorEncoder[BR] = 0;
while(nMotorEncoder[FL] < 700)
{
	motor[FL] = 25;
	motor[FR] = 25;
	motor[BL] = 25;
	motor[BR] = 25;
}
Stop();
servo[US] = 225;
WaitTime(2000);
GyroTurn(-90);
Stop();
FindRobotLeft();
FindRobotCenter();
FindRobotRight();
WaitTime(2000);
EncoderMove(17,30);
Stop();
FindRobotLeft();
FindRobotRight();
wait1Msec(500);
EncoderMove(17,30);
CollectSensorData();
Decision();
}
//---------------------[ FUNCTION 'ParkFunc1' ]-----------------------------------------

void ParkFunc1()
{

	writeDebugStreamLine("ParkFunc1");
	FindRobotLeft();
	IRServoTurn(90);
		EncoderMove(-10,30);
```

```
		Stop();
		wait1Msec(500);
		GyroTurn(40);
		Stop();
		wait1Msec(500);
		EncoderMove(25,30);
		wait1Msec(500);
		GyroTurn(-40);
		wait1Msec(500);
		//EncoderMove(5,25);
nMotorEncoder[FL] = 0;
nMotorEncoder[FR] = 0;
nMotorEncoder[BL] = 0;
nMotorEncoder[BR] = 0;

EncoderMove(36,100);

	Stop();
	EncoderMove(-36,100);
}
//---------------------[ FUNCTION 'ParkFunc2' ]-----------------------------------------

void ParkFunc2()
{
	writeDebugStreamLine("ParkFunc2");
	FindRobotLeft();
	IRServoTurn(90);
	EncoderMove(-4,15);
	GyroTurn(90);
	EncoderMove(36,45);
	GyroTurn(-125);
EncoderMove(45,100);

	Stop();
	EncoderMove(-10,100);
}
//---------------------[ FUNCTION 'ParkFunc3' ]-----------------------------------------

void ParkFunc3()
{
	writeDebugStreamLine("ParkFunc3");
	FindRobotLeft();
	IRServoTurn(90);
//EncoderMove(-4,15);
GyroTurn(87);

EncoderMove(30,50);
```

```
	Stop();
	GyroTurn(-90);
	_dirAC = HTIRS2readACDir(HTIRS2);
	while(_dirAC > 0)
	{
		_dirAC = HTIRS2readACDir(HTIRS2);
		motor[FL] = 30;
		motor[FR] = 30;
		motor[BL] = 30;
		motor[BR] = 30;
	}
EncoderMove(-8,30);
GyroTurn(-90);
	EncoderMove(13,40);
	EncoderMove(-15,30);
	GyroTurn(50);
	EncoderMove(22,30);
	GyroTurn(-35);
	EncoderMove(20,100);
	EncoderMove(-20,100);
}
//--------------------[ FUNCTION 'ParkDecision' ]-----------------------------------------

void ParkDecision()
{
	writeDebugStreamLine("RampDecision");
	if(_dirAC < 4 && _dirAC > 0)
	{
		writeDebugStreamLine("Position 1");
		IRtestResult = 1;
	}
	if(_dirAC == 4 || _dirAC == 5 || _dirAC == 6 || _dirAC == 8)
	{
		writeDebugStreamLine("Position 2");
		IRtestResult = 2;
	}
	if(_dirAC == 0)
	{
		writeDebugStreamLine("Position 3");
		IRtestResult = 3;
	}
	if(_dirAC < 0 || _dirAC > 9)
	{
	string errorTxt = "IR Fail";
	string errorTxtL2 = "Dir < 0 OR Dir > 9";
	Error(4, errorTxt);
```

```
		Error(5, errorTxtL2);
		wait1Msec(5000);
		}
		switch(IRtestResult)
		{
			case 1: ParkFunc1(); break;
			case 2: ParkFunc2(); break;
			case 3: ParkFunc3(); break;
		}
}

//---------------------[ FUNCTION 'BlueParking' ]-----------------------------------------

void BlueParking()
{
		writeDebugStreamLine("BlueParking");
		FloorFindRobotCenter();
		GyroTurn(-90);
		EncoderMove(12,30);
		GyroTurn(90);
		EncoderMove(44,30);
		FindRobotCenter();
		GyroTurn(90);
		EncoderMove(2,15);
		wait1Msec(2000);
		CollectSensorData();
		ParkDecision();
}
//---------------------[ FUNCTION 'RedParking' ]------------------------------------------

void RedParking()
{
		writeDebugStreamLine("RedParking");
		FloorFindRobotCenter();
		GyroTurn(-90);
		EncoderMove(10,30);
		GyroTurn(90);
		EncoderMove(30,30);
		FindRobotCenter();
		EncoderMove(14,30);
		GyroTurn(90);
		EncoderMove(2,15);
		wait1Msec(2000);
		CollectSensorData();
		ParkDecision();
}
//---------------------[ FUNCTION 'ColorDecision' ]---------------------------------------
```

```
void ColorDecision()
{

		raw1 = LSvalRaw(light);
		writeDebugStreamLine(" raw1: %d", raw1);
		EncoderMove(10,30);
		LSsetActive(light);

		Stop();
			LSsetActive(light);
			raw = LSvalRaw(light);
			writeDebugStreamLine(" raw: %d", raw);
			EncoderMove(2,30);
			LSsetActive(light);
			raw2 = LSvalRaw(light);
			writeDebugStreamLine(" raw2: %d", raw2);
		if(raw2 < 250 && raw2 > 230)
		{

			writeDebugStreamLine("Inside 1st Loop");
			LSsetActive(light);
			raw2 = LSvalRaw(light);
			writeDebugStreamLine(" raw2: %d", raw2);
			EncoderMove(5,25);
			LSsetActive(light);
			rawFloor = LSvalRaw(light);
			writeDebugStreamLine(" raw: %d", raw);
			if(raw1 < raw)
			{
				locationNumber = 4; //Red Parking Zone
				writeDebugStreamLine("RedParking");
			}
			if(raw1 > raw)
			{
				locationNumber = 3; //Blue Parking Zone
				writeDebugStreamLine("BlueParking");
			}
		}
		if(raw2 < 225 && raw2 > 200)
		{
			locationNumber = 1; //Blue Ramp
			writeDebugStreamLine("BlueRamp");
		}
		if(raw2 < 350 && raw2 > 335)
		{
			locationNumber = 2; //Red Ramp
```

```
                writeDebugStreamLine("RedRamp");
            }
            //if(raw || rawFloor < 200)
            //{
            //    locationNumber = 5;
            //}
        switch(locationNumber)
        {
            case 1: BlueRamp(); break;
            case 2: RedRamp(); break;
            case 3: BlueParking(); break;
            case 4: RedParking(); break;
            case 5: string errorTxtIR = "IR < 200"; Error(4,errorTxtIR); break;
}
}
```

```
//--------------------[ TASK MAIN ]-----------------------------------------------------------

task main()
        {
initializeRobot();

waitForStart();
        clearDebugStream();
        writeDebugStream("Start of Program");
//      nVolume = 4;
//      PlaySoundFile("HAL Start.rso");
//      while(bSoundActive)  // while a sound is actively playing:
//{
//  // do not continue until finished playing sound
//}
        servo[US] = 90;
        servo[TopVert] = 225;
        servo[TopHoriz] = 90;


ColorDecision();


}
```

```
//----------------------[ END OF PROGRAM ]-----------------------------------------------

//----------------------[ NOTES ]--------------------------------------------------------
// 1. Work on Defense Program
// 2. Work on Columns
//                  a. 30cm
//                  b. 60cm
//                  c. 90cm
// 3. Work on Tele-Op
//                  a. Integrate Case Statement
//----------------------[ REVISION HISTORY ]---------------------------------------------
//        REVISION DATE           : 12.23.2014
//        AUTHOR(S)                       :        Nikolaus Prusinski
//  DESCRIPTION    : All ramps and Parking Zones Complete
//                                                                                  :        Moving on to Defense Functions (See
Top of Program)
//        REVISION DATE           : 12.2.2014
//        AUTHOR(S)                       :        Nikolaus Prusinski
//  DESCRIPTION    : Established 3 functions with consistant reliability but decision in
//                                                                                  :        "Task Main" Requires more work for
accuracy.
//        REVISION DATE           : 10.14.2014
//        AUTHOR(S)                       :        Nikolaus Prusinski
//  DESCRIPTION    : Basic sturcture design as used in "RobotC Overview" implimented
//                                                                                  and began to copy necessary
commands/lines from previous years for
//                                                                                  SMUX sensors.
//-----------------------------------------------------------------------------------
//----------------------[ END OF FILE ]--------------------------------------------------
```