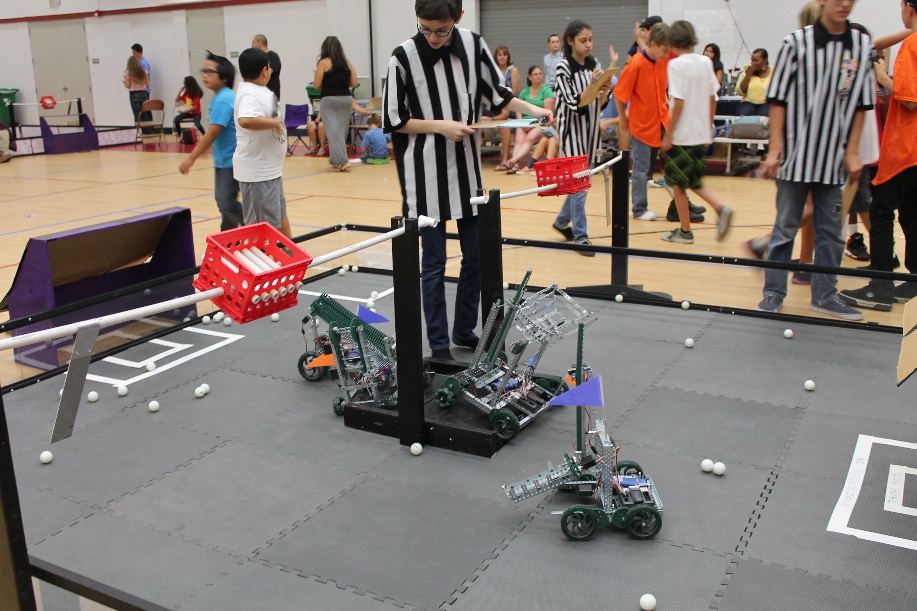
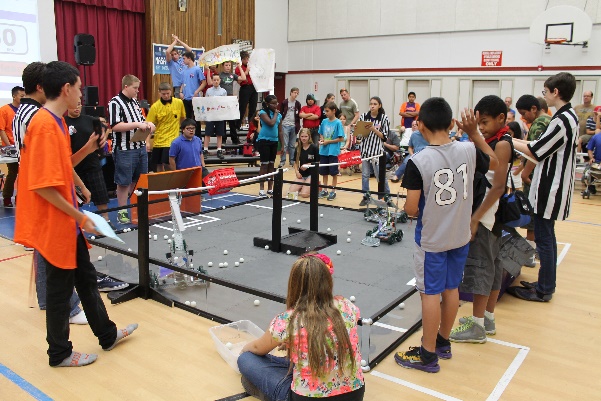
# OUTREACH

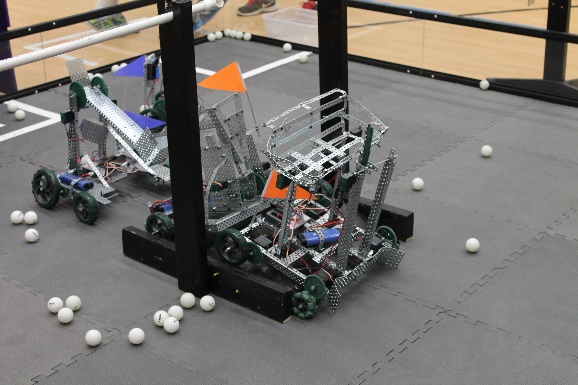




Community Outreach

VEX WORKSHOP

Each June, we host our biggest event of the year, a summer workshop in which 40 elementary- and middle-schoolers learn about science, technology, and robotics first-hand by building VEX robots. Ten teams consisting of six students were given a week to build a robot that could accomplish a simple game. High schoolers from local FTC and FRC robotics teams mentored them from June 23rd to the 28th.



At the end of each FTC-style match, both of the two-robot alliances earned points for racquetballs on the other alliance's side of the field. Toward the end of the week, teams participated in a mock competition to test their skills and show their parents what they learned. The teams then received awards for robot design, team spirit, and logo design.

Not only did the middle-schoolers build robots, but they also learn about FIRST core values such as Gracious Professionalism and becoming team players. During the week, presenters from NASA, Northrop Grumman, and Lockheed Martin came to tell the kids about their jobs and how robotics technology is used at their companies. The students especially enjoyed a presentation put on by NASA interns, which told them how science and engineering connects to the kids' daily lives. Teams also got to choose inventive names, create team posters, and design their own team buttons.

The other teams and companies that collaborated with us to make the event possible were Eagle robotics (FRC team #399 and FTC team #72), Bot Squad (Team #5011), Quartz Hill High school, the Robolopes (a local FRC team), Joe Walker Science Jets (several FLL teams), the NASA AERO Institute, Lockheed Martin, and Joe Walker Middle School. NASA provided the ten VEX kits used for the event.

ANTELOPE VALLEY FAIR

This past summer, our team reached out to the coordinators of our local county fair. We proposed an idea to have some of the FIRST teams in our valley get together and run a live display at the then upcoming fair. After several months of working with the other teams and the fair coordinators we had arranged for a few teams to bring their robots to the fair on the weekends and demonstrate them in the community building.

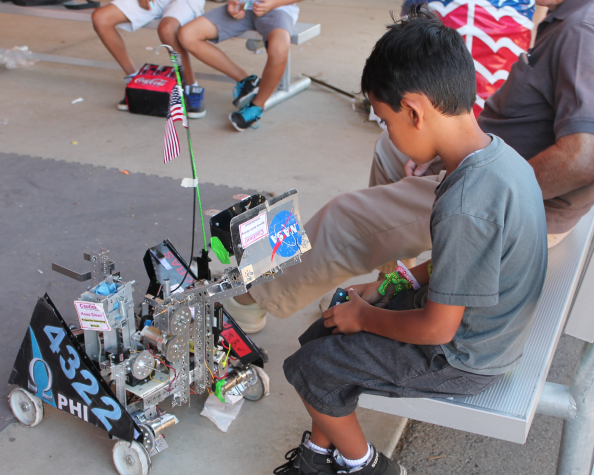
On the first day, we arrived early to work with one of the fair coordinators, Susan Chaisson, to set up our area. We had been allotted a small area inside the community building about 10ft by 20ft surrounded on three sides by chairs. This was a relatively small area to demonstrate a robot but we made it work. Along with the FRC Team 2339 Robolopes, we entertained crowds of people introducing them to robotics and teaching them about engineering.

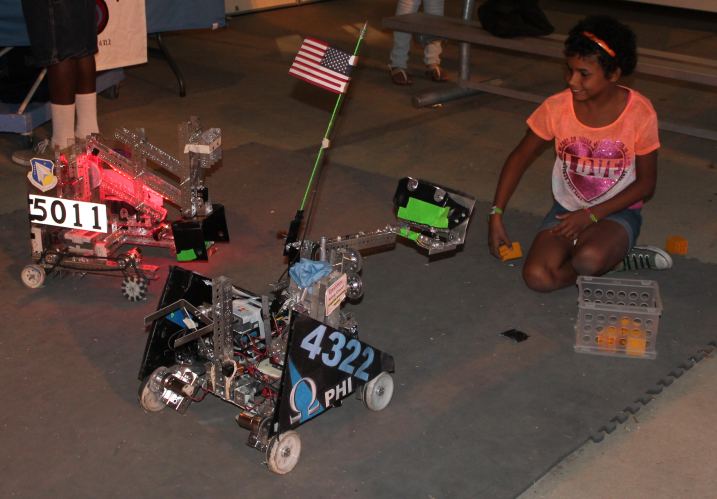
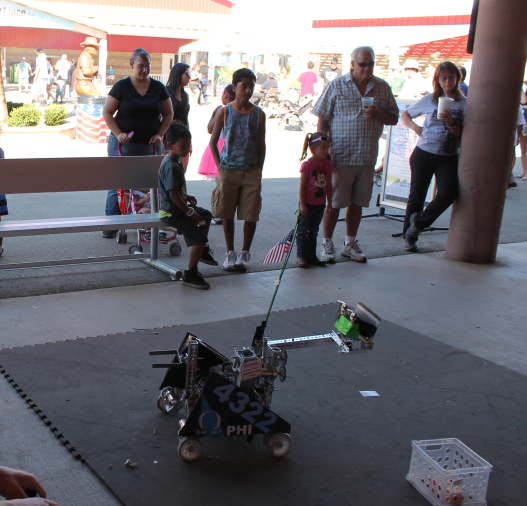
 

When Susan came back to check on us she said that there was a larger space in another part of the building that had just been cleared up for us. We then packed up our equipment and moved to the new location to continue demonstrating our robots. We also got to show some kids the internal workings of our robots.

When we returned the next weekend we were greeted with a surprise. Susan had talked to the rest of the fair board and they were so impressed at our success the previous weekend that they had given us the entire front patio of the community building to present our robots. With the extra space, we called in another FTC team, 5011 Bot Squad, and another FRC team, 5012 Griffin Gear. Our new location was right next to the front gate through which hundreds of people passed each hour. We also could now be more interactive with our audience by dropping our Star Trek Tribbles into their hands with our robot.

By the end of the fair we had handed out Tribbles and fliers to several hundred people and had encouraged several families to become involved in FIRST. The fair organizers where so impressed at our performance that they have invited us to come back again next year and they even promised us some display cases for a static display. The Antelope Valley Fair outreach has been by far our most successful and impacting outreach.

MENTOR HELP

Over the summer, we noticed that there were many kids that wanted to join robotics teams, but couldn’t due to a lack of open spaces, a lack of teams, or even a lack of mentors. We decided to help out by volunteering our knowledge at a mentor school.

THURSDAY NIGHT ON THE SQUARE

 Phi Robotics did a demonstration at Thursday Night on the Square- a local event where organizations can set up booths and put up displays of their organizations. The event runs from 5:30 to 8:00 PM on Thursday evenings in the summer. A few other local FTC teams, namely GarageBots and the Palmdale Aerospace Academy, also participated in the robotics display, as well as two FRC teams. The teams got the chance to share with visitors about robotics and explain what it is that teams do each year. These visitors ranged in age from young children to adults. They handed out FIRST flyers, explained the different age groups and their leagues. They also explained how the 2013-2014 game Block Party worked, and demonstrated how their robots from that season could pick up blocks and score them. Small LEGO robots were also there for children to drive. Both teams- Phi Alpha and Phi Omega- participated in the event and got the chance to show their robot to those who came in. Over three hundred people visited the robotics display and showed a lot of interest in the display.

**Appendix A – Distance from Acceleration trial Data**

Actually went 19”

**Appendix B – TeleOp**

**Tele-Op**

#pragma config(Hubs, S1, HTMotor, HTMotor, HTMotor, HTServo)

#pragma config(Sensor, S1, , sensorI2CMuxController)

#pragma config(Motor, mtr\_S1\_C1\_1, rightDrive, tmotorTetrix, openLoop, reversed)

#pragma config(Motor, mtr\_S1\_C1\_2, leftDrive, tmotorTetrix, openLoop)

#pragma config(Motor, mtr\_S1\_C2\_1, rightLift, tmotorTetrix, openLoop)

#pragma config(Motor, mtr\_S1\_C2\_2, leftLift, tmotorTetrix, openLoop)

#pragma config(Motor, mtr\_S1\_C3\_1, intake, tmotorTetrix, openLoop, reversed)

#pragma config(Motor, mtr\_S1\_C3\_2, tilt, tmotorTetrix, openLoop)

#pragma config(Servo, srvo\_S1\_C4\_1, release, tServoStandard)

#pragma config(Servo, srvo\_S1\_C4\_2, servo2, tServoNone)

#pragma config(Servo, srvo\_S1\_C4\_3, servo3, tServoNone)

#pragma config(Servo, srvo\_S1\_C4\_4, servo4, tServoNone)

#pragma config(Servo, srvo\_S1\_C4\_5, servo5, tServoNone)

#pragma config(Servo, srvo\_S1\_C4\_6, servo6, tServoNone)

//\*!!Code automatically generated by 'ROBOTC' configuration wizard !!\*//

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

//

// Tele-Operation Mode Code Template

//

// This file contains a template for simplified creation of an tele-op program for an FTC

// competition.

//

// You need to customize two functions with code unique to your specific robot.

//

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

#include "JoystickDriver.c" //Include file to "handle" the Bluetooth messages.

#include "Buttonmap.h"

int deadband = 10;

bool driving = false;

float driveFraction = 1.0;

task Drive;

task TiltM;

task Buttons;

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

//

// initializeRobot

//

// Prior to the start of tele-op mode, you may want to perform some initialization on your robot

// and the variables within your program.

//

// In most cases, you may not have to add any code to this function and it will remain "empty".

//

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

void initializeRobot()

{

return;

}

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

//

// Main Task

//

// The following is the main code for the tele-op robot operation. Customize as appropriate for

// your specific robot.

//

// Game controller / joystick information is sent periodically (about every 50 milliseconds) from

// the FMS (Field Management System) to the robot. Most tele-op programs will follow the following

// logic:

// 1. Loop forever repeating the following actions:

// 2. Get the latest game controller / joystick settings that have been received from the PC.

// 3. Perform appropriate actions based on the joystick + buttons settings. This is usually a

// simple action:

// \* Joystick values are usually directly translated into power levels for a motor or

// position of a servo.

// \* Buttons are usually used to start/stop a motor or cause a servo to move to a specific

// position.

// 4. Repeat the loop.

//

// Your program needs to continuously loop because you need to continuously respond to changes in

// the game controller settings.

//

// At the end of the tele-op period, the FMS will autonmatically abort (stop) execution of the program.

//

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

task Drive()

{

//Define storage variables

int rawX, rawY;

float shapedX, shapedY;

float leftPower, rightPower;

while (true)

{

//Get joystick data

getJoystickSettings(joystick);

//Check if the joystick is being used on any axis

if (abs(joystick.joy1\_x1) > deadband)

rawX = joystick.joy1\_x1 - (sgn(joystick.joy1\_x1) \* deadband); //Set variables and subtract deadband from total to avoid jump

else

rawX = 0;

if (abs(joystick.joy1\_y1) > deadband)

rawY = joystick.joy1\_y1 - (sgn(joystick.joy1\_y1) \* deadband); //Set variables and subtract deadband from total to avoid jump

else

rawY = 0;

//Shape the powers based on equation: y = (2/225)(x\*abs(x))+(14/45)x

shapedX = 0.0088888888888889 \* (rawX \* abs(rawX)) + 0.3111111111111111 \* rawX;

shapedY = 0.0088888888888889 \* (rawY \* abs(rawY)) + 0.3111111111111111 \* rawY;

//shapedX = (float)(2/225) \* (rawX \* abs(rawX)) + (float)(14/45) \* rawX;

//shapedY = (float)(2/225) \* (rawY \* abs(rawY)) + (float)(14/45) \* rawY;

//Determine power of each side by slowing one side and increasing the other appropriately

leftPower = shapedY + shapedX/2;

rightPower = shapedY - shapedX/2;

//Finally, set the power to the motors

motor[leftDrive] = leftPower \* driveFraction;

motor[rightDrive] = rightPower \* driveFraction;

if (abs(leftPower) > 0 || abs(rightPower) > 0)

driving = true;

else

driving = false;

}

}

task TiltM()

{

//Define storage variables

int rawY;

float shapedY;

float power;

while (true)

{

//Get joystick data

getJoystickSettings(joystick);

//Check if the joystick is being used on any axis

if (abs(joystick.joy1\_y2) > deadband)

rawY = joystick.joy1\_y2 - (sgn(joystick.joy1\_y2) \* deadband); //Set variables and subtract deadband from total to avoid jump

else

rawY = 0;

//Shape the powers based on equation: y = (2/225)(x\*abs(x))+(14/45)x

shapedY = 0.0088888888888889 \* (rawY \* abs(rawY)) + 0.3111111111111111 \* rawY;

//shapedY = (float)(2/225) \* (rawY \* abs(rawY)) + (float)(14/45) \* rawY;

//Determine power of each side by slowing one side and increasing the other appropriately

power = shapedY;

//Finally, set the power to the motors

motor[tilt] = power / 4;

}

}

task Buttons()

{

bool open = false;

while (true)

{

getJoystickSettings(joystick);

if (open)

{

if (servo[release] > 100)

servo[release] = 0;

else

servo[release] = 256;

driveFraction = 1.0;

open = false;

}

if (joy1Btn(BTN\_RB))

open = true;

else

open = false;

if (joy1Btn(BTN\_A))

{

motor[intake] = -100;

}

else if (joy1Btn(BTN\_LB))

{

motor[intake] = 100;

}

else

{

motor[intake] = 0;

}

if (joy1Btn(BTN\_LT))

{

motor[leftLift] = -50;

motor[rightLift] = -50;

if (servo[release] < 100)

open = true;

}

else if (joy1Btn(BTN\_RT))

{

motor[leftLift] = 50;

motor[rightLift] = 50;

if (!driving)

{

driveFraction = 0.25;

}

}

else

{

motor[leftLift] = 0;

motor[rightLift] = 0;

}

}

}

task main()

{

initializeRobot();

waitForStart(); // wait for start of tele-op phase

StartTask(Drive);

StartTask(TiltM);

StartTask(Buttons);

while (true)

{

}

}

**Appendix C – Autonomous Functions**

**Move**

//Move Function

void Move(float Distance, float Speed)

{

float countsPerInch = 20;

float userSpeed = 0.0, distTrav = 0.0, time = 0.0, powerShiftConst = 0.3, speedPercent = 0.0;

float leftPower = 0.0, rightPower = 0.0, leftSpeed = 0.0, rightSpeed = 0.0;

long leftEncodCount = 0.0, rightEncodCount = 0.0, prevLeftEncodCount = 0.0, prevRightEncodCount = 0.0;

int direction = sgn(Distance);

Distance = abs(Distance);

Distance \*= countsPerInch;

Speed \*= countsPerInch;

userSpeed = Speed;

leftPower = 5;

rightPower = 5;

nMotorEncoder[rightRear] = 0;

nMotorEncoder[leftRear] = 0;

ClearTimer(T1);

motor[leftFront] = leftPower;

motor[leftRear] = leftPower;

motor[rightFront] = rightPower;

motor[rightRear] = rightPower;

float timeElapsed = 0.0;

float iterations = 0;

Speed = 0;

while (distTrav < Distance)

{

time = time1[T1] / 1000;

leftEncodCount = nMotorEncoder[leftRear];

rightEncodCount = nMotorEncoder[rightRear];

/\*if (iterations < 1.0)

{

timeElapsed += time;

if (timeElapsed > iterations)

{

Speed += 0.25;

iterations += 0.2;

}

}

else

Speed = userSpeed;\*/

speedPercent = -1 \* log(distTrav/Distance);

if (speedPercent > 1)

speedPercent = 1;

Speed \*= speedPercent;

distTrav = (leftEncodCount + rightEncodCount) / 2;

leftSpeed = (leftEncodCount - prevLeftEncodCount) / time;

rightSpeed = (rightEncodCount - prevRightEncodCount) / time;

if (leftSpeed == 0)

leftSpeed = Speed;

if (rightSpeed == 0)

rightSpeed = Speed;

leftPower += (Speed - leftSpeed) \* powerShiftConst;

rightPower += (Speed - rightSpeed) \* powerShiftConst;

if (abs(leftPower) > 100)

leftPower = 100 \* sgn(leftPower);

if (abs(rightPower) > 100)

rightPower = 100 \* sgn(rightPower);

motor[leftFront] = direction \* leftPower;

motor[leftRear] = direction \* leftPower;

motor[rightFront] = direction \* rightPower;

motor[rightRear] = direction \* rightPower;

prevLeftEncodCount = nMotorEncoder[leftRear];

prevRightEncodCount = nMotorEncoder[rightRear];

ClearTimer(T1);

writeDebugStreamLine("left: %d", leftEncodCount);

writeDebugStreamLine("right: %d", rightEncodCount);

writeDebugStreamLine("distance: %d", Distance);

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

}

motor[leftFront] = 0;

motor[leftRear] = 0;

motor[rightFront] = 0;

motor[rightRear] = 0;

}

**Turn**

void Turn( float turnDegrees )

{

// Store new bias

GyroBias = ((float)avgdata / 20);

int state = 2;

int time;

int time\_old = 0;

int delta\_time;

float maxVelocity = 180.0; // max angular velocity in deg/sec

float GyroOld;

float GyroNew;

int rotDirection = 1;

int motorSpeed = 0;

float DegreeGain = 5.0;

float degreesToGo;

int MAX\_GYRO\_SPEED = 85; //maximum motor speed allowed in turns

int MIN\_GYRO\_SPEED = 40; //minimum motor speed allowed in turns

float Angle;

// Determine the rotation direction. Set to negative for clock-wise rotation.

if( turnDegrees < 0 ) rotDirection = -1;

// Set the Angle and components to zero befor turning

Angle = 0.0;

degreesToGo = 0.0;

time\_old = nPgmTime;

GyroOld = 0.0;

wait1Msec(100);

// remove the backlash and freeplay from the motors before zeroing the encoders by commanding

// a very low power for a short time

motorSpeed = 5;

motor[B\_L] = -rotDirection \* motorSpeed;

motor[B\_R] = rotDirection \* motorSpeed;

wait1Msec(10);

// reset the back motor encoders to zero

nMotorEncoder[B\_R] = 0;

nMotorEncoder[B\_L] = 0;

ClearTimer(T3);

//---------------------------------------------------------------------------

// Move the motors until the gyro Angle indicates the rotation is complete

//---------------------------------------------------------------------------

while( ( abs(turnDegrees) - abs(Angle) ) > 3 )

{

time = nPgmTime; // this timer wraps around

delta\_time = abs(time - time\_old); // delta time in ms

if (delta\_time < 1) // protect against divide by zero

{

delta\_time = 1;

}

// read the gyro sensor minus the bias offset. GyroBias must be declared and

// computed in the calling program.

GyroNew = -((float)SensorValue[Gyro] - GyroBias);

// limit the gyro to the max achievable by the bot to minimize data spikes.

if (abs(GyroNew) > maxVelocity) GyroNew = sgn(GyroNew)\*maxVelocity;

// deadband for the gyro to eliminate drift due to noise

if (abs(GyroNew) <= 0.2) GyroNew = 0.0;

// compute the integral of the angular rate using a trapazoidal approximation

// http://en.wikipedia.org/wiki/Numerical\_integration

Angle = Angle + 0.001 \* (float)delta\_time \* 0.5 \*(GyroNew + GyroOld);

// update the old values for the next time through the loop

time\_old = time;

GyroOld = GyroNew;

// Calculate the rotation remaining

degreesToGo = abs(turnDegrees) - abs(Angle);

// motor speed is proportional to the amount of rotation remaining

motorSpeed = (int)(DegreeGain \* abs(degreesToGo));

// limit the motor speed to be greater than 15 and less than 75

if (abs(motorSpeed) > MAX\_GYRO\_SPEED) motorSpeed = MAX\_GYRO\_SPEED;

if (abs(motorSpeed) < MIN\_GYRO\_SPEED) motorSpeed = MIN\_GYRO\_SPEED;

motor[B\_L] = rotDirection \* motorSpeed ; // looking from the top, left side moves back, right

motor[B\_R] = -rotDirection \* motorSpeed ; // side moves forward for (+) CCW rotation

if( time100[T3] > 50 ) break; // if the bot is not done turning in 5 sec, quit

wait1Msec(15); // wait 20 ms to allow a reasonable period for the integration

} // End while rotation remaining

// since we have now reached the target rotation (the while loop is done), stop all drive motors

motor[B\_R] = 0;

motor[B\_L] = 0;

// reset the back motor encoders to zero

nMotorEncoder[B\_R] = 0;

nMotorEncoder[B\_L] = 0;

} // End of Turn\_Gyro()

**Distance from Acceleration**

void Move(float distance, float power)

{

float offset = 0;

int xRaw, yRaw, zRaw = 0;

float xAccel = 0;

float prevXAccel = 0;

float prevXRaw = 0;

int timeOld = 0;//, timeNew = 0;

float deltaTime = 0;

float velocity = 0;

float prevVelocity = 0;

float dist = 0;

//float mPowers = 0;

power = abs(power) \* sgn(distance);

distance = abs(distance);

float accelBias = 0;

for (int i = 0; i < 20; i++)

{

HTACreadAllAxes(accel, xRaw, yRaw, zRaw);

accelBias += sqrt((xRaw \* xRaw)+(yRaw \* yRaw)+(zRaw \* zRaw));

//accelBias += xRaw;

}

offset = accelBias/20;

clearDebugStream();

ClearTimer(T1);

//timeNew = nPgmTime;

int rampUpCount = 300\*0.05;

while (true)

{

if (abs(dist) < distance)

{

if (rampUpCount <= 300)

rampUpCount += 1;

motor[rightDrive] = power \* rampUpCount/100;

motor[leftDrive] = power \* rampUpCount/100;

HTACreadAllAxes(accel, xRaw, yRaw, zRaw);

xAccel = (sqrt((xRaw \* xRaw)+(yRaw \* yRaw)+(zRaw \* zRaw)-(offset\*offset)) / 200.0) \* 386.09;

//xAccel = (((float)xRaw - (float)offset) / 200.0) \* 386.09;

//writeDebugStreamLine("%f",xAccel);

deltaTime = time1[T1] \* 0.001;

//deltaTime = (timeNew - timeOld) \* 0.001;

if (abs(xAccel) <= 0.1)

xAccel = 0;

velocity += deltaTime/2 \* (prevXAccel + xAccel);

dist += deltaTime/2 \* (prevVelocity + velocity);

//writeDebugStreamLine("Accel: %f", xAccel);

ClearTimer(T1);

//timeOld = timeNew;

prevXAccel = xAccel;

prevVelocity = velocity;

//wait1Msec(10);

/\*prevXRaw = xRaw;

do

{

HTACreadAllAxes(accel, xRaw, yRaw, zRaw);

}while(xRaw == prevXRaw);\*/

}

else

{

motor[rightDrive] = 0;

motor[leftDrive] = 0;

}

}

}

**I2C Master Bit Banging Tests**

#pragma config(Sensor, S1, , sensorI2CCustom)

//\*!!Code automatically generated by 'ROBOTC' configuration wizard !!\*//

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

//

// Autonomous Mode Code Template

//

// This file contains a template for simplified creation of an autonomous program for an TETRIX robot

// competition.

//

// You need to customize two functions with code unique to your specific robot.

//

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

#include "JoystickDriver.c" //Include file to "handle" the Bluetooth messages.

#include "common.h"

#define clock 0x02 //B1

#define data 0x01 //B0

bool scl = false;

bool sda = false;

byte buffer[10];

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

//

// initializeRobot

//

// Prior to the start of autonomous mode, you may want to perform some initialization on your robot.

// Things that might be performed during initialization include:

// 1. Move motors and servos to a preset position.

// 2. Some sensor types take a short while to reach stable values during which time it is best that

// robot is not moving. For example, gyro sensor needs a few seconds to obtain the background

// "bias" value.

//

// In many cases, you may not have to add any code to this function and it will remain "empty".

//

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

void initializeRobot()

{

// Place code here to sinitialize servos to starting positions.

// Sensors are automatically configured and setup by ROBOTC. They may need a brief time to stabilize.

return;

}

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

//

// Main Task

//

// The following is the main code for the autonomous robot operation. Customize as appropriate for

// your specific robot.

//

// The types of things you might do during the autonomous phase (for the 2008-9 FTC competition)

// are:

//

// 1. Have the robot follow a line on the game field until it reaches one of the puck storage

// areas.

// 2. Load pucks into the robot from the storage bin.

// 3. Stop the robot and wait for autonomous phase to end.

//

// This simple template does nothing except play a periodic tone every few seconds.

//

// At the end of the autonomous period, the FMS will autonmatically abort (stop) execution of the program.

//

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

//General I2C Communication Functions

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void writeLines()

{

scl = !scl;

sda = !sda;

tByteArray msg;

msg[0] = 3;

msg[1] = 0x10;

msg[2] = 0x4E;

msg[3] = (scl && sda) ? (clock + data) : (scl ? clock : (sda ? data : 0x00));

writeI2C(S1, msg);

scl = !scl;

sda = !sda;

msg[2] = 0x4D;

msg[3] = (scl && sda) ? (clock + data) : (scl ? clock : (sda ? data : 0x00));

writeI2C(S1, msg);

}

void clockHi()

{

scl = true;

writeLines();

}

void clockLo()

{

scl = false;

writeLines();

}

bool checkClock()

{

tByteArray msg;

msg[0] = 2;

msg[1] = 0x10;

msg[2] = 0x4C;

tByteArray reply;

writeI2C(S1, msg, reply, 1);

if ((reply[0] & clock) > 0)

return true;

else

return false;

}

void dataHi()

{

sda = true;

writeLines();

}

void dataLo()

{

sda = false;

writeLines();

}

bool checkData()

{

tByteArray msg;

msg[0] = 2;

msg[1] = 0x10;

msg[2] = 0x4C;

tByteArray reply;

writeI2C(S1, msg, reply, 1);

if ((reply[0] & data) > 0)

return true;

else

return false;

}

void delay()

{

int \_random = 0;

for (int i = 0; i <= 5; i++)

\_random += 1;

}

void i2cStart()

{

dataHi();

delay();

clockHi();

delay();

dataLo();

delay();

clockLo();

delay();

}

void i2cStop()

{

clockLo();

delay();

dataLo();

delay();

clockHi();

delay();

dataHi();

delay();

}

void i2cTx(ubyte d)

{

for (int x = 8; x > 0; x--)

{

if (d & 0x80)

dataHi();

else

dataLo();

delay();

clockHi();

d <<= 1;

clockLo();

}

delay();

dataHi();

clockHi();

while (!checkClock()){}

//Ack comes here

delay();

clockLo();

}

void writeData(ubyte a, ubyte iA, ubyte d)

{

a <<= 1;

i2cStart();

i2cTx(a);

i2cTx(iA);

i2cTx(d);

i2cStop();

}

byte i2cRx(bool ack)

{

ubyte d = 0;

dataHi();

for(int x = 0; x < 8; x++)

{

d <<= 1;

clockHi();

while(!checkClock()){}

delay();

bool t = checkData();

d |= (int)t;

writeDebugStreamLine("%d", t);

clockLo();

}

if (ack)

dataLo();

else

dataHi();

clockHi();

delay();

clockLo();

dataHi();

return d;

}

void readData(ubyte a, ubyte iA, int bytesToRead)

{

a <<= 1;

i2cStart();

i2cTx(a);

i2cTx(iA);

i2cStart();

i2cTx(a + 1);

int last = 0;

for (int i = 0; i < bytesToRead - 1; i++)

{

buffer[i] = i2cRx(true);

last = i + 1;

}

buffer[last] = i2cRx(false);

i2cStop();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Sensor Specific Functions

#define accelAddr 0x53

#define powerCtrlAddr 0x2D

#define dataFormatAddr 0x31

#define xAddr0 0x32

#define yAddr0 0x34

#define zAddr0 0x36

void setupAccel()

{

writeData(accelAddr, dataFormatAddr, 0x01);

writeData(accelAddr, dataFormatAddr, 0x08);

}

int xAccel = 0.0;

int yAccel = 0.0;

int zAccel = 0.0;

void updateAcceleration()

{

readData(accelAddr, xAddr0, 6);

xAccel = (((int)buffer[0]) << 8) | buffer[1];

yAccel = (((int)buffer[2]) << 8) | buffer[3];

zAccel = (((int)buffer[4]) << 8) | buffer[5];

}

task main()

{

initializeRobot();

//waitForStart(); // Wait for the beginning of autonomous phase.

clearDebugStream();

setupProtoBoard();

setupAccel();

bool t = false;

while (true)

{

updateAcceleration();

writeDebugStreamLine("X Accel: %f", checkData());

/\*if (t)

dataHi();

else

dataLo();

t = !t;

sleep(3000);\*/

}

}

**Appendix D – Autonomous Program**

**Autonomy**

#pragma config(Hubs, S1, HTServo, HTMotor, HTMotor, HTMotor)

#pragma config(Sensor, S4, ir, sensorHiTechnicIRSeeker1200)

#pragma config(Motor, mtr\_S1\_C2\_1, rightFront, tmotorTetrix, openLoop)

#pragma config(Motor, mtr\_S1\_C2\_2, rightRear, tmotorTetrix, openLoop, encoder)

#pragma config(Motor, mtr\_S1\_C3\_1, lift, tmotorTetrix, openLoop)

#pragma config(Motor, mtr\_S1\_C3\_2, intake, tmotorTetrix, openLoop)

#pragma config(Motor, mtr\_S1\_C4\_1, leftFront, tmotorTetrix, openLoop, reversed)

#pragma config(Motor, mtr\_S1\_C4\_2, leftRear, tmotorTetrix, openLoop, reversed, encoder)

#pragma config(Servo, srvo\_S1\_C1\_1, output, tServoStandard)

#pragma config(Servo, srvo\_S1\_C1\_2, hook1, tServoStandard)

#pragma config(Servo, srvo\_S1\_C1\_3, hook2, tServoNone)

#pragma config(Servo, srvo\_S1\_C1\_4, servo4, tServoNone)

#pragma config(Servo, srvo\_S1\_C1\_5, servo5, tServoNone)

#pragma config(Servo, srvo\_S1\_C1\_6, servo6, tServoNone)

//\*!!Code automatically generated by 'ROBOTC' configuration wizard !!\*//

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

//

// Autonomous Mode Code Template

//

// This file contains a template for simplified creation of an autonomous program for an TETRIX robot

// competition.

//

// You need to customize two functions with code unique to your specific robot.

//

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

#include "JoystickDriver.c" //Include file to "handle" the Bluetooth messages.

#include "newMove.c"

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

//

// initializeRobot

//

// Prior to the start of autonomous mode, you may want to perform some initialization on your robot.

// Things that might be performed during initialization include:

// 1. Move motors and servos to a preset position.

// 2. Some sensor types take a short while to reach stable values during which time it is best that

// robot is not moving. For example, gyro sensor needs a few seconds to obtain the background

// "bias" value.

//

// In many cases, you may not have to add any code to this function and it will remain "empty".

//

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

void initializeRobot()

{

// Place code here to sinitialize servos to starting positions.

// Sensors are automatically configured and setup by ROBOTC. They may need a brief time to stabilize.

return;

}

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

//

// Main Task

//

// The following is the main code for the autonomous robot operation. Customize as appropriate for

// your specific robot.

//

// The types of things you might do during the autonomous phase (for the 2008-9 FTC competition)

// are:

//

// 1. Have the robot follow a line on the game field until it reaches one of the puck storage

// areas.

// 2. Load pucks into the robot from the storage bin.

// 3. Stop the robot and wait for autonomous phase to end.

//

// This simple template does nothing except play a periodic tone every few seconds.

//

// At the end of the autonomous period, the FMS will autonmatically abort (stop) execution of the program.

//

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

task main()

{

initializeRobot();

//waitForStart(); // Wait for the beginning of autonomous phase.

clearDebugStream();

Move(-77, 0.7);

Servo[hook1] = 255;

Servo[hook2] = 0;

Turn(30);

Move(80, 1);

}