/\*

Author: Alec Korver

Date: 3/25/2017

Description:

Principia College ASME Tank Competition Arduino Mega 2560 Shield

The following code is for controlling (2) track motors and (1 or 2) tennis

ball launcher motors.

Each shield uses

(1) Custom PCB Shield

(2) Pololu VNH5019 motor drivers (5V - 24V) 12A Cont. 30A Peak 20kHz max PWM

(1) Turnigy 9x8C V2 Receiver - 8 Channel

(1) Pololu 5V S18V20F5 Voltage Regulator

To control the motors you will need (1) Turnigy 9x 8 Channel handheld

remote control. The controll will need to be synced with the reciever prior

to use. Only 3 channels are initially used for this system. There are 5

additional channels, which can be used for future expansions.

Handheld remote control info

https://hobbyking.com/media/file/725056143X2037269X20.pdf

//Good information on reading the PWM signals from the reciever

http://www.impulseadventure.com/elec/attiny-spi-rc-pwm.html

IMPORTANT NOTE ABOUT LAUNCHER

The launcher motor should be up to full speed before a tennis ball is dropped in.

This ensures that it will work similar to a pitching machine. Otherwise the launcher will be

underpowered. To increase the firing distance of the launcher there are a few options:

1)Increase the wheel inertia (increase mass of wheel)

2)Increase diameter of the launcher wheel (Larger diameter means higher speed at edge)

3)Both 1 and 2

4)Increase the voltage to the motor (Do not run continuously at higher voltage or the motor WILL burn up!!)

\*/

//CODE FOR CUSTOM CONTRLOL

//Pins for Pololu PWM motor driver 1

//The first 4 pins need to be set either high or low

//How they are set determines whether the motor will be

//rotating CW or CCW or if it is braking or regenerating.

//See datasheet on motor driver for more details.

const int M1InA\_pin = 22;

const int M1EnA\_pin = 24;

const int M1InB\_pin = 26;

const int M1EnB\_pin = 28;

const int M1Pwm\_pin = 5;

//Pins for Pololu PWM motor driver 2

//The first 4 pins need to be set either high or low (1 or 0)

//How they are set determines whether the motor will be

//rotating CW or CCW or if it is braking or regenerating.

//See datasheet on motor driver for more details.

const int M2InA\_pin = 23;

const int M2EnA\_pin = 25;

const int M2InB\_pin = 27;

const int M2EnB\_pin = 29;

const int M2Pwm\_pin = 6;

//Pins for Turnigy 9x8C V2 Reciever

//Each pin maps to one channel of the receiver

//The input from the reciever to each pin will be a PWM signal

//The width of the PWM signal is set by the hand held remote control

//Checking this width tells us the different thumbstick and switch positions

const int Ch1\_pin = 2;

const int Ch2\_pin = 3;

const int Ch3\_pin = 18;

const int Ch4\_pin = 19;

const int Ch5\_pin = 4;

//Variables to store the PWM values read from each receiver channel(pin)

long Ch1Pwm;

long Ch2Pwm;

long Ch3Pwm;

long Ch4Pwm;

long Ch5Pwm;

//Maximum pulse width in microseconds - TET AND SET THESE VALUES!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

long Ch1MaxPw;

long Ch2MaxPw;

long Ch3MaxPw;

long Ch4MaxPw;

long Ch5MaxPw;

//Middle value for pulse width - This is the center position for the thumbsticks - TEST AND SET THESE VALUES!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

int Ch1MidPw;

int Ch2MidPw;

int Ch3MidPw;

int Ch4MidPw;

int Ch5MidPw;

//Variables for holding the PWM value (0 - 255) that we will send to the

//motor driver. This will set our left and right motor speeds

//255 is 100% output aka FULL SPEEED. FULL 24V.

int M1\_Pwm;

int M2\_Pwm;

void setup() {

// put your setup code here, to run once:

Serial.begin(9600);

pinMode(M1InA\_pin,OUTPUT);

pinMode(M1EnA\_pin,OUTPUT);

pinMode(M1InB\_pin,OUTPUT);

pinMode(M1EnB\_pin,OUTPUT);

pinMode(M1Pwm\_pin,OUTPUT);

pinMode(M2InA\_pin,OUTPUT);

pinMode(M2EnA\_pin,OUTPUT);

pinMode(M2InB\_pin,OUTPUT);

pinMode(M2EnB\_pin,OUTPUT);

pinMode(M2Pwm\_pin,OUTPUT);

pinMode(Ch1\_pin,INPUT);

pinMode(Ch2\_pin,INPUT);

pinMode(Ch3\_pin,INPUT);

pinMode(Ch4\_pin,INPUT);

pinMode(Ch5\_pin,INPUT);

//Sets the motor driver to forward and reverse functions only

digitalWrite(M1EnA\_pin,HIGH);

digitalWrite(M1EnB\_pin,HIGH);

digitalWrite(M2EnA\_pin,HIGH);

digitalWrite(M2EnB\_pin,HIGH);

//Sets motor1 and motor2 to forward direction on startup

digitalWrite(M1InA\_pin,HIGH);

digitalWrite(M1InB\_pin,LOW);

digitalWrite(M2InA\_pin,HIGH);

digitalWrite(M2InB\_pin,LOW);

}

//MAIN LOOP - THIS WILL BE RUNNING AND LOOPING CONTINUOUSLY

void loop() {

// put your main code here, to run repeatedly:

//OTHER CHANNELS MAY BE ENABLED HERE IF YOU ARE ADDING OTHER FUNCTIONALITY

//In Mode 1 Elevator is left track and Throttle is right track

//In Mode 2 these settings are reversed

//Channel1Data(); //Uncomment to use Channel 1

Channel2Data(); //Checks Motor 1 throttle setting - AKA ELEVATOR

Channel3Data(); //Checks Motor 2 throttle setting - AKA THROTTLE

//Channel4(Data); //Uncomment to use Channel 4

Channel5Data(); //Checks the launcher motor switch

MotorControl(); //This sets the new speed and direction for the track motors and turns on/off the launcher motors.

}

//CUSTOM FUNCTIONS - ALL CODE BELOW HERE IS CUSTOM FUNCTIONS WHICH

//ARE CALLED DURING THE PROGRAM OPERATION

//Sets the direction of Motor1 to Forward

void Motor1Forward(){

digitalWrite(M1InA\_pin,HIGH);

digitalWrite(M1InB\_pin,LOW);

analogWrite(M1Pwm\_Pin,M1\_Pwm);

}

//Sets the direction of Motor2 to Forward

void Motor2Forward(){

digitalWrite(M2InA\_pin,HIGH);

digitalWrite(M2InB\_pin,LOW);

analogWrite(M2Pwm\_Pin,M2\_Pwm);

}

//Sets the direction of Motor1 to Reverse

void Motor1Reverse(){

digitalWrite(M1InA\_pin,LOW);

digitalWrite(M1InB\_pin,HIGH);

analogWrite(M1Pwm\_Pin,M1\_Pwm);

}

//Sets the direction of Motor2 to Reverse

void Motor2Reverse(){

digitalWrite(M2InA\_pin,LOW);

digitalWrite(M2InB\_pin,HIGH);

analogWrite(M2Pwm\_Pin,M2\_Pwm);

}

//Function to read the PWM signal from Channel 1

void Channel1Data(){

bool Ch1CurrentState;

bool Ch1PreviousState;

long Ch1RisingEdgeTime;

long Ch1FallingEdgeTime;

long Ch1TimeDif;

Ch1CurrentState = digitalRead(Ch1\_pin);

//Triggers on rising edge

if(Ch1CurrentState == HIGH and Ch1PreviousState == LOW){

Ch1PreviousState = Ch1CurrentState;

Ch1RisingEdgeTime = micros();

}

//Triggers on falling edge

else if(Ch1CurrentState == LOW and Ch1PreviousState == HIGH){

Ch1PreviousState = Ch1CurrentState;

Ch1FallingEdgeTime = micros();

Ch1TimeDif = Ch1FallingEdgeTime - Ch1RisingEdgeTime;

Ch1Pwm = Ch1TimeDif;

}

//Signal is still in HIGH state

//else if(Ch1CurrentState == HIGH and Ch1PreviousState == HIGH){

//}

//Signal is still in LOW state

//else if(Ch1CurrentState == LOW and Ch1PreviousState == LOW){

//}

//Any other conditions - No conditions should exist here

else{

}

}

//Function to read the PWM signal from Channel 2

void Channel2Data(){

bool Ch2CurrentState;

bool Ch2PreviousState;

long Ch2RisingEdgeTime;

long Ch2FallingEdgeTime;

long Ch2TimeDif;

Ch2CurrentState = digitalRead(Ch2\_pin);

//Triggers on rising edge

if(Ch2CurrentState == HIGH and Ch2PreviousState == LOW){

Ch2PreviousState = Ch2CurrentState;

Ch2RisingEdgeTime = micros();

}

//Triggers on falling edge

else if(Ch2CurrentState == LOW and Ch2PreviousState == HIGH){

Ch2PreviousState = Ch2CurrentState;

Ch2FallingEdgeTime = micros();

Ch2TimeDif = Ch2FallingEdgeTime - Ch2RisingEdgeTime;

Ch2Pwm = Ch2TimeDif;

}

//Signal is still in HIGH state

//else if(Ch2CurrentState == HIGH and Ch2PreviousState == HIGH){

//}

//Signal is still in LOW state

//else if(Ch2CurrentState == LOW and Ch2PreviousState == LOW){

//}

//Any other conditions - No conditions should exist here

else{

}

}

//Function to read the PWM signal from Channel 3

void Channel3Data(){

bool Ch3CurrentState;

bool Ch3PreviousState;

long Ch3RisingEdgeTime;

long Ch3FallingEdgeTime;

long Ch3TimeDif;

Ch3CurrentState = digitalRead(Ch3\_pin);

//Triggers on rising edge

if(Ch3CurrentState == HIGH and Ch3PreviousState == LOW){

Ch3PreviousState = Ch3CurrentState;

Ch3RisingEdgeTime = micros();

}

//Triggers on falling edge

else if(Ch3CurrentState == LOW and Ch3PreviousState == HIGH){

Ch3PreviousState = Ch3CurrentState;

Ch3FallingEdgeTime = micros();

Ch3TimeDif = Ch3FallingEdgeTime - Ch3RisingEdgeTime;

Ch3Pwm = Ch3TimeDif;

}

//Signal is still in HIGH state

//else if(Ch3CurrentState == HIGH and Ch3PreviousState == HIGH){

//}

//Signal is still in LOW state

//else if(Ch3CurrentState == LOW and Ch3PreviousState == LOW){

//}

//Any other conditions - No conditions should exist here

else{

}

}

//Function to read the PWM signal from Channel 4

void Channel4Data(){

bool Ch4CurrentState;

bool Ch4PreviousState;

long Ch4RisingEdgeTime;

long Ch4FallingEdgeTime;

long Ch4TimeDif;

Ch4CurrentState = digitalRead(Ch4\_pin);

//Triggers on rising edge

if(Ch4CurrentState == HIGH and Ch4PreviousState == LOW){

Ch4PreviousState = Ch4CurrentState;

Ch4RisingEdgeTime = micros();

}

//Triggers on falling edge

else if(Ch4CurrentState == LOW and Ch4PreviousState == HIGH){

Ch4PreviousState = Ch4CurrentState;

Ch4FallingEdgeTime = micros();

Ch4TimeDif = Ch4FallingEdgeTime - Ch4RisingEdgeTime;

Ch4Pwm = Ch4TimeDif;

}

//Signal is still in HIGH state

//else if(Ch4CurrentState == HIGH and Ch4PreviousState == HIGH){

//}

//Signal is still in LOW state

//else if(Ch4CurrentState == LOW and Ch4PreviousState == LOW){

//}

//Any other conditions - No conditions should exist here

else{

}

}

//Function to read the PWM signal from Channel 5

void Channel5Data(){

bool Ch5CurrentState;

bool Ch5PreviousState;

long Ch5RisingEdgeTime;

long Ch5FallingEdgeTime;

long Ch5TimeDif;

Ch5CurrentState = digitalRead(Ch5\_pin);

//Triggers on rising edge

if(Ch5CurrentState == HIGH and Ch5PreviousState == LOW){

Ch5PreviousState = Ch5CurrentState;

Ch5RisingEdgeTime = micros();

}

//Triggers on falling edge

else if(Ch5CurrentState == LOW and Ch5PreviousState == HIGH){

Ch5PreviousState = Ch5CurrentState;

Ch5FallingEdgeTime = micros();

Ch5TimeDif = Ch5FallingEdgeTime - Ch5RisingEdgeTime;

Ch5Pwm = Ch5TimeDif;

}

//Signal is still in HIGH state

//else if(Ch5CurrentState == HIGH and Ch5PreviousState == HIGH){

//}

//Signal is still in LOW state

//else if(Ch5CurrentState == LOW and Ch5PreviousState == LOW){

//}

//Any other conditions - No conditions should exist here

else{

}

}

void MotorControl(){

//Motor 1

//If thumbstick is down

if(Ch2Pwm < Ch2MidPw){

M1\_Pwm = (255/Ch2MidPw) \* (Ch2MidPw - Ch2Pwm);

Serial.print("\n")

Serial.print("M1 Reverse: ");

Serial.print(M1\_Pwm);

Motor1Reverse();

}

//If thumbstick is in the middle

else if (Ch2Pwm == Ch2MidPw){

M1\_Pwm = 0;

analogWrite(M1Pwm\_Pin,M1\_Pwm);

}

//If thumbstick is up

else if (Ch2Pwm > Ch2MidPw){

M1\_Pwm = (255/Ch2MidPw) \* (Ch2Pwm - Ch2MidPw);

Serial.print("\n")

Serial.print("M1 Forward: ");

Serial.print(M1\_Pwm);

Motor1Forward();

}

else{

}

//Motor 2

//If thumbstick is down

if(Ch3Pwm < Ch3MidPw){

M2\_Pwm = (255/Ch3MidPw) \* (Ch3MidPw - Ch3Pwm);

Serial.print("\n")

Serial.print("M2 Reverse: ");

Serial.print(M2\_Pwm);

Motor2Reverse();

}

//If thumbstick is in the middle

else if (Ch3Pwm == Ch3MidPw){

M2\_Pwm = 0;

analogWrite(M2Pwm\_Pin,M2\_Pwm);

}

//If thumbstick is up

else if (Ch3Pwm > Ch3MidPw){

M2\_Pwm = (255/Ch3MidPw) \* (Ch3Pwm - Ch3MidPw);

Serial.print("\n")

Serial.print("M2 Forward: ");

Serial.print(M2\_Pwm);

Motor2Forward();

}

else{

}

//Launcher Motor Relay!!!!!!!!!!!!!!!!!!!!!!!!!!!!WRITE CODE AND ASSIGN PIN FOR RELAY CONTROL

}