Appendix C

Source Code

**Main.ino**

//20725728

//<Includes>

#include <SD.h>

#include <SPI.h>

#include <Wire.h>

#include <LSM303.h>

#include <Time.h>

#include <math.h>

#include "MyStructures.h"

#include "MyHeaders.h"

//</Includes>

//<Global Variables>

int state = 1; //1-wait for GPS; 2-Navigate; 3-Standyby

int Manual\_Control = 0; // 1=manual Control enabled;

int Debug = 1;

bool runProgram = true;

int steercounter=0;

/\* D13 - S2 (manual/auto)

\* D8 - S3 (green)

\* D4 - S1 (Power)

\* D12 - led on

\* D11 - led active

\* D9 - led M

\* D7 - led GPs

\* D6 - led A

\*/

int manualS = 13;

int compassS = 8;

int powerS = 4;

int LED\_ON = 12;

int LED\_ACTIVE = 11;

int LED\_MAN = 9;

int LED\_GPS = 7;

int LED\_AUTO = 6;

//Navigation ----------------------------------

int throttle\_down\_distance = 7; //(m) distance at which the vessel reduces below full throttle speed.

int full\_steer\_angle = 20;//(degrees) the angle beyond which the vessel will use full steering.

int throttle\_percentage = 100; //percentage full throttle applied (full throttle doesnt seem to work)

int steering\_percentage = 100; //percentage steering lock applied.

int arrival\_threshold = 7; //distance at which it is considered that the vessel reached the point.

//GPS ------------------------------------------

GPSrx pos;

bool GPS\_connected = false;

char validGPS = 'V';

int targetIndex = 0;

//PWM ------------------------------------------

float RC = 84000;

int POT\_LH\_input = A1;

int POT\_RH\_input = A0;

PWMControl rightSide;

PWMControl leftSide;

//SDCard -----------------------------------------

// CS pin is pin 53,

const int chipSelect = 53; //Set CS interface

File root;

String fileName = "datalog.txt";

//SD Read --------------------------------------

GPSpt points[20];

//Compass --------------------------------------

LSM303 compass;

LSM303::vector<int16\_t> running\_min = {-517, -1212, -1124}, running\_max = {2672, 1956, 1895};

int calibrate = 1; //0 - calibrate compass 1- set compass limits 2 - operating mode

//Debug ----------------------------------------

int addPointsCnt = 0;

//</Global Variables>

void setup(){

pwmPin3();

pwmPin5();

Serial.begin(9600);

Serial1.begin(9600);

Wire.begin();

compass.init();

compass.enableDefault();

initialize\_SD();

SetupIOs();

readGPSPoints();

compass.m\_min = (LSM303::vector<int16\_t>){running\_min.x, running\_min.y, running\_min.z};

compass.m\_max = (LSM303::vector<int16\_t>){running\_max.x, running\_max.y, running\_max.z};

//receiveAllGPSPoints();

}

void loop() {

double test;

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

SwitchesAndLEDs();

if(calibrate == 0){

CalibrateCompass();

}else{

if(runProgram){

if(Manual\_Control == 1){

updatePWM();

writeThrottle();

//printThrottle();

receiveGPSdata();

}else{

switch(state){

case 1:

test = GetHeading();

//Serial.println(test);

receiveGPSdata();

Serial.print("Waiting for GPS\n");

if(GPS\_connected){

state = 2;

}

break;

case 2:

receiveGPSdata();

navigate();

delay(100);

break;

case 3:

delay(500);

break;

}

}

}

delay(200);

}

}

void Listen(){

int byteRX = Serial.read();

if(byteRX != -1){

switch(byteRX){

case 'A'://switch to automatic control

Manual\_Control = 0;

break;

case 'M'://switch to manual control

Manual\_Control = 1;

break;

}

}

}

void SetupIOs(){

pinMode(manualS, INPUT);

pinMode(compassS, INPUT);

pinMode(compassS, INPUT);

pinMode(LED\_ON, OUTPUT);

pinMode(LED\_ACTIVE, OUTPUT);

pinMode(LED\_MAN, OUTPUT);

pinMode(LED\_GPS, OUTPUT);

pinMode(LED\_AUTO, OUTPUT);

/\* D13 - S2 (manual/auto)

\* D8 - S3 (compass calibration)

\* D4 - S1 (Power)

\* D12 - led on

\* D11 - led active

\* D9 - led M

\* D7 - led GPs

\* D6 - led A

\*/

}

void SwitchesAndLEDs(){

if(digitalRead(compassS)){

calibrate = 0;

}else{

if(calibrate == 0){

calibrate = 1;

}

}

if(digitalRead(powerS)){

digitalWrite(LED\_ON, HIGH);

runProgram = true;

}else{//stop the main loop running and set the motors to neutral

digitalWrite(LED\_ON, LOW);

runProgram = false;

leftSide.duty\_Cycle = 75;

rightSide.duty\_Cycle = 75;

power\_thrusters();

}

if(digitalRead(manualS)){

digitalWrite(LED\_MAN, HIGH);

digitalWrite(LED\_AUTO, LOW);

Manual\_Control = 1;

}else{

digitalWrite(LED\_MAN, LOW);

digitalWrite(LED\_AUTO, HIGH);

Manual\_Control = 0;

}

if(digitalRead(LED\_ACTIVE)){

digitalWrite(LED\_ACTIVE, LOW);

}else{

digitalWrite(LED\_ACTIVE, HIGH);

}

if(GPS\_connected){

digitalWrite(LED\_GPS,HIGH);

}else{

digitalWrite(LED\_GPS,LOW);

}

}

**AutoControl.ino**

void navigate(){

double distance;

double bearing\_error;

double target\_bearing;

double maxBearing;

double minBearing;

GPSpt target = points[targetIndex];

distance = GetDistance(pos.latitudeDecimalDegrees, pos.longitudeDecimalDegrees, target.latitudeDecimalDegrees, target.longitudeDecimalDegrees);

target\_bearing = GetBearing(pos.latitudeDecimalDegrees, pos.longitudeDecimalDegrees, target.latitudeDecimalDegrees, target.longitudeDecimalDegrees);

double bearingE\_1 = target\_bearing - pos.course;

double bearingE\_2 = abs(target\_bearing - pos.course) - 360;

if(abs(bearingE\_1) < abs(bearingE\_2)){

bearing\_error = bearingE\_1;

}else{

bearing\_error = bearingE\_2;

}

if(distance < throttle\_down\_distance){

throttle\_percentage = distance/(2\*throttle\_down\_distance);

if(distance < arrival\_threshold){

NextPoint();

}

}else{

throttle\_percentage = 95;

}

if(abs(bearing\_error) < full\_steer\_angle){

steering\_percentage = ((float)abs(bearing\_error)/(float)full\_steer\_angle) \* 100;

}else{

steering\_percentage = 100;

}

maxBearing = max(pos.course, target\_bearing);

minBearing = min(pos.course, target\_bearing);

if((maxBearing == target\_bearing) ^ ((maxBearing-minBearing)>180)){

writeError(distance, bearing\_error, target\_bearing, "RIGHT");

steer\_right();

}else{

steer\_left();

writeError(distance, bearing\_error, target\_bearing, "LEFT");

}

power\_thrusters();

if(Debug == 1){

writeError(distance, bearing\_error, target\_bearing, "debug");

}else{

writeNavigation(distance, bearing\_error, target\_bearing);

}

writeThrottle();

}

void steer\_left(){

int throttle\_difference = 2\*throttle\_percentage;

rightSide.duty\_Cycle = 75 + (25\*((float)throttle\_percentage/100));

leftSide.duty\_Cycle = rightSide.duty\_Cycle - (25 \* ((throttle\_difference \* ((float)steering\_percentage/100))/100));

if(rightSide.duty\_Cycle >=100){

rightSide.duty\_Cycle = 95;

}

if(leftSide.duty\_Cycle >= 100){

leftSide.duty\_Cycle = 95;

}

}

void steer\_right(){

int throttle\_difference = 2\*throttle\_percentage;

leftSide.duty\_Cycle = 75 + (25\*((float)throttle\_percentage/100));

rightSide.duty\_Cycle = leftSide.duty\_Cycle - (25\* ((throttle\_difference \* ((float)steering\_percentage/100))/100));

if(rightSide.duty\_Cycle >=100){

rightSide.duty\_Cycle = 95;

}

if(leftSide.duty\_Cycle >= 100){

leftSide.duty\_Cycle = 95;

}

}

**Compass.ino**

void CalibrateCompass(){

char report[80];

compass.read();

running\_min.x = min(running\_min.x, compass.m.x);

running\_min.y = min(running\_min.y, compass.m.y);

running\_min.z = min(running\_min.z, compass.m.z);

running\_max.x = max(running\_max.x, compass.m.x);

running\_max.y = max(running\_max.y, compass.m.y);

running\_max.z = max(running\_max.z, compass.m.z);

snprintf(report, sizeof(report), "min: {%+6d, %+6d, %+6d} max: {%+6d, %+6d, %+6d}",

running\_min.x, running\_min.y, running\_min.z,

running\_max.x, running\_max.y, running\_max.z);

Serial.println(report);

}

double GetHeading(){

if(calibrate = 1){

compass.m\_min = (LSM303::vector<int16\_t>){running\_min.x, running\_min.y, running\_min.z};

compass.m\_max = (LSM303::vector<int16\_t>){running\_max.x, running\_max.y, running\_max.z};

calibrate = 2;

}

char report[80];

compass.read();

Serial.print("Reading Heading");

// snprintf(report, sizeof(report), "A: %6d %6d %6d M: %6d %6d %6d",

// compass.a.x, compass.a.y, compass.a.z,

// compass.m.x, compass.m.y, compass.m.z);

Serial.println(compass.heading());

return compass.heading();

}

**Debug.ino**

void writeError(double dist, float bearing\_error, float target\_bearing, String dir){

String navigationErrors ="Target\nLat#"+String(points[targetIndex].latitudeDecimalDegrees,8)+"#Long#"+String(points[targetIndex].longitudeDecimalDegrees,8)+"#Bearing#"+String(target\_bearing,6)+

"\nCurrent:\nLat#"+String(pos.latitudeDecimalDegrees,8)+"#Long#"+String(pos.longitudeDecimalDegrees,8)+"#Bearing#\t"+String(pos.course,6)+

"\nErrors:\nDist#"+String(dist,6)+"#Bearing\_Error#"+String(bearing\_error,6)+

"\n\n"+dir+"\t"+String(bearing\_error,6)+"\n\n";

Serial.print(navigationErrors);

}

void receiveGPSPoint(){

int rx;

points[addPointsCnt].latitudeDecimalDegrees = 0;

points[addPointsCnt].longitudeDecimalDegrees = 0;

for(int i =0;i<7;i++){

do{

rx = Serial.read();

}while(rx==-1);

points[addPointsCnt].latitudeDecimalDegrees += (rx-48)\*(pow(10,(1-i)));

}

points[addPointsCnt].latitudeDecimalDegrees \*= -1;

do{

rx = Serial.read();

}while(rx==-1);

for(int i =0;i<7;i++){

do{

rx = Serial.read();

}while(rx==-1);

points[addPointsCnt].longitudeDecimalDegrees += (rx-48)\*(pow(10,(1-i)));

}

Serial.println("\tLAT: "+String(points[addPointsCnt].latitudeDecimalDegrees,8)+"\tLONG: "+String(points[addPointsCnt].longitudeDecimalDegrees,8));

}

void receiveAllGPSPoints(){

int rx;

Serial.print("How many points are to be entered? :\n");

do{

rx = Serial.read();

}while(rx==-1);

int numPoints = rx - 48;

for(int i = 0;i<numPoints;i++){

Serial.print("\nEnter point "+String(i)+":\n");

receiveGPSPoint();

}

}

void printThrottle(){

Serial.println("LEFT#"+String(leftSide.analog)+"#Right#"+String(rightSide.analog));

}

**DirectionCalculator.ino**

double GetBearing(double latitude1, double longitude1, double latitude2, double longitude2){

double phi1 = latitude1 \* (PI/180);

double phi2 = latitude2 \* (PI/180);

double delta\_phi = (latitude2 - latitude1) \* (PI/180);

double delta\_lambda = (longitude2 - longitude1) \* (PI/180);

double theta = (double)atan2( sin(delta\_lambda)\*cos(phi2), cos(phi1)\*sin(phi2) - sin(phi1)\*cos(phi2)\*cos(delta\_lambda));

double bearing = ((theta \* 180/PI));

if(bearing<0){

return bearing+360;

}

return bearing;

}

//return the distance between two GPS points in meters

double GetDistance(double latitude1, double longitude1, double latitude2, double longitude2){

double radius = 6.371E6;

double phi1 = latitude1 \* (PI/180);

double phi2 = latitude2 \* (PI/180);

double delta\_phi = (latitude2 - latitude1) \* (PI/180);

double delta\_lambda = (longitude2 - longitude1) \* (PI/180);

double a = sq(sin(delta\_phi/2)) + cos(phi1) \* cos(phi2) \* sq(sin(delta\_lambda/2));

double c = 2 \* atan2(sqrt(a), sqrt(1-a));

double dist = radius \* c;

return dist;

}

void NextPoint(){

points[targetIndex].passed = true;

Serial.print("\nPoint passed\n");

targetIndex ++;

Serial.println("Target Index :"+String(targetIndex));

if(targetIndex >=20 || points[targetIndex].orderPos <=0){

state = 3;

}

}

**GPS.ino**

void saveGPSPoints(){

if(validGPS == 'A'){

String output = "Time#"+String(pos.date)+" "+String(pos.UTC)+"#DD#Lat#"+String(pos.latitudeDecimalDegrees,8)+"#Long#"+

String(pos.longitudeDecimalDegrees,8)+

"#DMS#Lat#"+String((long)pos.latitude)+"#Long#"+String((long)pos.longitude)+"#Speed#"+String(pos.knots)+"#Bearing#"+String(pos.course);

File dataFile = SD.open("datalog.txt", FILE\_WRITE);

if (dataFile) {

dataFile.println(output);

Serial.println(output);

dataFile.close();

// dataString outputs to the serial port

}

// If the file cannot be opened, the serial port outputs an error message

else {

Serial.println("error opening datalog");

}

}

}

void writeHeader(){

String header = "GPS data "+String(pos.date)+"\_"+String(pos.UTC)+".txt";

File dataFile = SD.open("datalog.txt", FILE\_WRITE);

dataFile.close();

if(SD.exists("datalog.txt")){

Serial.write("File Does exists\n");

dataFile = SD.open("datalog.txt", FILE\_WRITE);

}else{

Serial.write("File Does NOT exist\n");

}

if(dataFile){

dataFile.println(header);

}else{

Serial.println("Error opening datalog for header");

}

dataFile.close();

//Throttle Header

header = "Throttle\_data "+String(pos.date)+"\_"+String(pos.UTC)+".txt";

dataFile = SD.open("throttle.txt", FILE\_WRITE);

dataFile.close();

if(SD.exists("throttle.txt")){

Serial.write("File Does exists\n");

dataFile = SD.open("throttle.txt", FILE\_WRITE);

}else{

Serial.write("File Does NOT exist\n");

}

if(dataFile){

dataFile.println(header);

}else{

Serial.println("Error opening datalog for throttle header");

}

dataFile.close();

//Nav Header

header = "Nav\_data "+String(pos.date)+"\_"+String(pos.UTC)+".txt";

dataFile = SD.open("nav.txt", FILE\_WRITE);

dataFile.close();

if(SD.exists("nav.txt")){

Serial.write("File Does exists\n");

dataFile = SD.open("nav.txt", FILE\_WRITE);

}else{

Serial.write("File Does NOT exist\n");

}

if(dataFile){

dataFile.println(header);

}else{

Serial.println("Error opening datalog for nav header");

}

dataFile.close();

}

void writeThrottle(){

String throttleState = "<LEFT <DC "+String(leftSide.duty\_Cycle)+" /DC><Input "+String(leftSide.analog)+" /Input><us "+String(RC\*leftSide.duty\_Cycle) + " /us>/LEFT>\t";

throttleState += "<RIGHT <DC "+String(rightSide.duty\_Cycle)+" /DC><Input "+String(rightSide.analog)+" /Input><us "+String(RC\*rightSide.duty\_Cycle) + " /us>/RIGHT>";

File dataFile = SD.open("throttle.txt", FILE\_WRITE);

if(dataFile){

dataFile.println(throttleState);

}else{

Serial.println("Error opening throttle file.");

}

dataFile.close();

}

void writeNavigation(double dist, double bearing\_error, double target\_bearing){

String navigationErrors ="Target#Lat#"+String(points[targetIndex].latitudeDecimalDegrees,8)+"#Long#"+String(points[targetIndex].longitudeDecimalDegrees,8)+"#Bearing#"+String(target\_bearing,6)+

"#Current:#Lat#"+String(pos.latitudeDecimalDegrees,8)+"#Long#"+String(pos.longitudeDecimalDegrees,8)+"#Bearing#"+String(pos.course,6)+

"#Errors:#Dist#"+String(dist,6)+"#Bearing\_Error#"+String(bearing\_error,6)+

"#Motors:#Throttle%#"+String(throttle\_percentage)+"#Steering%#"+String(steering\_percentage)+"#Left\_DC#"+String(leftSide.duty\_Cycle,5)+"#Right\_DC#"+String(rightSide.duty\_Cycle,5)+"\n\n\n";

File dataFile = SD.open("nav.txt",FILE\_WRITE);

if(dataFile){

dataFile.println(navigationErrors);

}else{

Serial.println("Error opening nav error file");

}

dataFile.close();

}

MyStructures.h

struct GPSpt{

int orderPos;

long latitude;

double latitudeDecimalDegrees = 0;

long longitude;

double longitudeDecimalDegrees = 0;

bool passed = false;

};

struct GPSrx {

int date;

int UTC;

long latitude;

double latitudeDecimalDegrees;

long longitude;

double longitudeDecimalDegrees;

double knots;

double course;

};

struct PWMControl {

int analog = 505; //the middle for both of the POTs

float duty\_Cycle = 75; //75% is neutral.

};

**PWMSetup.ino**

void updatePWM() {

Right\_Throttle();

Left\_Throttle();

// Serial.write("LEFT: ");

// Serial.print(leftSide.analog);

// Serial.write(" RIGHT: ");

// Serial.print(rightSide.analog);

// Serial.println();

// Serial.write("LEFT: ");

// Serial.print(leftSide.duty\_Cycle);

// Serial.write(" RIGHT: ");

// Serial.print(rightSide.duty\_Cycle);

// Serial.println();

}

void power\_thrusters(){

TC2->TC\_CHANNEL[1].TC\_RA = (RC\*rightSide.duty\_Cycle)/100;

TC2->TC\_CHANNEL[0].TC\_RA = (RC\*leftSide.duty\_Cycle)/100;

}

/\*\*

left \* right

max ~655 \* max ~720

min ~285 \* min ~352

mid~480 \* mid ~538

440 - 505 stop \* 485 - 572 stop

\*/

void Right\_Throttle(){

float analog\_Ratio = 0;

float difference = 0;

rightSide.analog = analogRead(POT\_RH\_input);

if(rightSide.analog > 720){ // Full Forward

rightSide.duty\_Cycle = 95;

}else if(rightSide.analog >565){ //Forward

analog\_Ratio = (float)(720 - rightSide.analog)/(float)(720 - 565);

difference = (float)(95 - 75) \* analog\_Ratio;

rightSide.duty\_Cycle = 95 - difference;

}else if(rightSide.analog < 355){ // Full Reverse

rightSide.duty\_Cycle = 50;

}else if(rightSide.analog < 500){ //Reverse

analog\_Ratio = (float)(500 - rightSide.analog)/(float)(500-355);

difference = (float)(75-50) \* analog\_Ratio;

rightSide.duty\_Cycle = 75 - difference;

}else{ //Neutral

rightSide.duty\_Cycle = 75;

}

TC2->TC\_CHANNEL[1].TC\_RA = (RC\*rightSide.duty\_Cycle)/100;

}

void Left\_Throttle(){

float analog\_Ratio = 0;

float difference = 0;

leftSide.analog = analogRead(POT\_LH\_input);

if(leftSide.analog > 655){ // Full Forward

leftSide.duty\_Cycle = 95;

}else if(leftSide.analog >505){ //Forward

analog\_Ratio = (float)(655 - leftSide.analog)/(float)(665 - 505);

difference = (float)(95 - 75) \* analog\_Ratio;

leftSide.duty\_Cycle = 95 - difference;

}else if(leftSide.analog < 285){ // Full Reverse

leftSide.duty\_Cycle = 50;

}else if(leftSide.analog < 440){ //Reverse

analog\_Ratio = (float)(440 - leftSide.analog)/(float)(440-285);

difference = (float)(75-50) \* analog\_Ratio;

leftSide.duty\_Cycle = 75 - difference;

}else{ //Neutral

leftSide.duty\_Cycle = 75;

}

TC2->TC\_CHANNEL[0].TC\_RA = (RC\*leftSide.duty\_Cycle)/100;

}

void pwmPin3() {

PMC->PMC\_PCER1 |= PMC\_PCER1\_PID34; // TC7 power ON - Timer Counter 2 channel 1 IS TC7 - See page 38

PIOC->PIO\_PDR |= PIO\_PDR\_P28; // The pin is no more driven by GPIO

PIOC->PIO\_ABSR |= PIO\_PC28B\_TIOA7; // Periperal type B - See page 859

TC2->TC\_CHANNEL[1].TC\_CMR = TC\_CMR\_TCCLKS\_TIMER\_CLOCK1 // MCK/2, clk on rising edge

| TC\_CMR\_WAVE // Waveform mode

| TC\_CMR\_WAVSEL\_UP\_RC // UP mode with automatic trigger on RC Compare

| TC\_CMR\_ACPA\_CLEAR // Clear TIOA7 on RA compare match -- See page 883

| TC\_CMR\_ACPC\_SET; // Set TIOA7 on RC compare match

/// Mck = 84 MHz

TC2->TC\_CHANNEL[1].TC\_RC = RC; //<\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Frequency = (Mck/2)/TC\_RC Hz = 500 Hz

TC2->TC\_CHANNEL[1].TC\_RA = 63000; //<\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Duty cycle = (TC\_RA/TC\_RC) \* 100

TC2->TC\_CHANNEL[1].TC\_IER = TC\_IER\_CPCS; // Interrupt on RC compare match

NVIC\_EnableIRQ(TC7\_IRQn);

TC2->TC\_CHANNEL[1].TC\_CCR = TC\_CCR\_SWTRG | TC\_CCR\_CLKEN; // Software trigger TC7 counter and enable

}

void pwmPin5() {

/\*Set power management controller register

\* PMC - power controller register

\* PMC\_PCER1 - Peripheral clock enable register 1 Pg 563

\* PMC\_PCER1\_PID33 - Enable peripheral clock PID33 - Timer Channel 6

\*/

PMC->PMC\_PCER1 |= PMC\_PCER1\_PID33; // TC6 power ON - Timer Counter 2 channel 2 IS TC6 - See page 38

/\*Disable pin input/output

\* PIOC - parallel input output controller C

\* PIO\_PDR - PIO Disable Register Pg 634

\* PIO\_PDR\_P25 - Disable P25 (PC25 = TIOA6)

\* PIO\_ABSR - Peripheral AB select register Pg 656

\* PIO\_PC25B\_TIOA6 - set PC25 = peripheral B function (TIOA6)

\*/

PIOC->PIO\_PDR |= PIO\_PDR\_P25; // The pin is no more driven by GPIO

PIOC->PIO\_ABSR |= PIO\_PC25B\_TIOA6; // Periperal type B - See page 859

/\* Set Timer counter settings TC UI Pg 879

\* TC\_CMR - Channel mode register Pg 883

\* TC\_CMR\_TCCLKS\_TIMER\_CLOCK1 - Set timer clock 1 (Mck/2) & counter incremented on rising edge

\* TC\_CMR\_WAVE - Enable waveform mode

\* TC\_CMR\_WAVSEL\_UP\_RC - UP mode with automatic trigger on RC compare

\* TC\_CMR\_ACPA\_CLEAR - Clear effect on TIOA on RA compare

\* TC\_CMR\_ACPC\_SET - Set effect on TIOA on RC compare

\*

\* TC\_RC - Register C Pg 891

\* TC\_RA - Register A Pg 889

\*

\* TC\_IER - Interrupt enable register Pg 894

\* TC\_IER\_CPCS - enable interrupt on RC Compare match

\*

\* TC\_CCR - Channel Control Register Pg 880

\* TC\_CCR\_SWTRG - A software trigger is performed: the counter is reset and the clock is started.

\* TC\_CCR\_CLKEN - enable clock

\*/

TC2->TC\_CHANNEL[0].TC\_CMR = TC\_CMR\_TCCLKS\_TIMER\_CLOCK1 // MCK/2, clk on rising edge

| TC\_CMR\_WAVE // Waveform mode

| TC\_CMR\_WAVSEL\_UP\_RC // UP mode with automatic trigger on RC Compare

| TC\_CMR\_ACPA\_CLEAR // Clear TIOA6 on RA compare match -- See page 883

| TC\_CMR\_ACPC\_SET; // Set TIOA6 on RC compare match

/// Mck = 84 MHz

TC2->TC\_CHANNEL[0].TC\_RC = RC; //<\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Frequency = (Mck/2)/TC\_RC Hz = 500Hz

TC2->TC\_CHANNEL[0].TC\_RA = 63000; //<\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Duty cycle = (TC\_RA/TC\_RC) \* 100

TC2->TC\_CHANNEL[0].TC\_IER = TC\_IER\_CPCS; // Interrupt on RC compare match

NVIC\_EnableIRQ(TC6\_IRQn);

TC2->TC\_CHANNEL[0].TC\_CCR = TC\_CCR\_SWTRG | TC\_CCR\_CLKEN; // Software trigger TC6 counter and enable

}

void TC6\_Handler() {

/\*Interrupt triggered on RC compare match

\* TC\_SR - Status Register Pg - 892

\* Read the status registe to clear the RC compare and RA compare match flags.

\*/

TC2->TC\_CHANNEL[0].TC\_SR;

}

void TC7\_Handler() {

TC2->TC\_CHANNEL[1].TC\_SR;

}

**SDCard.ino**

void initialize\_SD(){

Serial.print("Initializing SD card..."); //Serial port output data

pinMode(10, OUTPUT);

if (!SD.begin(chipSelect)) { //If communication from the CS port fails with the SD card, the serial port outputs information: Card failed, or not present

Serial.println("Card failed, or not present");

return;

}

Serial.println("card initialized."); //SD card communication success, serial output card information initialized.

//root = SD.open("/");

//printDirectory(root, 0);

//writeHeader();

}

void printDirectory(File dir, int numTabs) {

while (true) {

File entry = dir.openNextFile();

if (! entry) {

// no more files

break;

}

for (uint8\_t i = 0; i < numTabs; i++) {

Serial.print('\t');

}

Serial.print(entry.name());

if (entry.isDirectory()) {

Serial.println("/");

printDirectory(entry, numTabs + 1);

} else {

// files have sizes, directories do not

Serial.print("\t\t");

Serial.println(entry.size(), DEC);

}

entry.close();

}

}

**SDRead.ino**

void readGPSPoints(){

File gpsFile = SD.open("points.txt");

char pointsBuf[540];

if(gpsFile){

int fileLength = gpsFile.available();

gpsFile.read(pointsBuf, fileLength);

Serial.println(pointsBuf);

Serial.println("fileLength: "+String(fileLength));

decodeGPSBuf(pointsBuf, (fileLength - 20)/22);

}else{

Serial.println("Error getting gpsPoints");

}

}

void decodeGPSBuf(char buf[], int numberPoints){

int buffpos = 0;

for(int i = 0;i<numberPoints;i++){

buffpos = i\*22;

Serial.println(buf[buffpos]);

points[i].orderPos = buf[buffpos]\*10 + buf[buffpos+1];

buffpos+=2;

points[i].latitudeDecimalDegrees = (double) ((double)charToInt(buf, buffpos, buffpos+8) /100000);

buffpos+=8;

if(buf[buffpos] == 'S'){

points[i].latitudeDecimalDegrees \*= -1;

}

buffpos+=1;

points[i].longitudeDecimalDegrees = (double) ((double)charToInt(buf, buffpos, buffpos+8) /100000);

buffpos+=8;

if( buf[buffpos] == 'W'){

points[i].longitudeDecimalDegrees \*= -1;

}

Serial.println("Point "+String(i)+": "+String(points[i].latitudeDecimalDegrees,7)+"\t"+String(points[i].longitudeDecimalDegrees,7));

}

}

**SDWrite.ino**

void saveGPSPoints(){

if(validGPS == 'A'){

String output = "Time#"+String(pos.date)+" "+String(pos.UTC)+"#DD#Lat#"+String(pos.latitudeDecimalDegrees,8)+"#Long#"+

String(pos.longitudeDecimalDegrees,8)+

"#DMS#Lat#"+String((long)pos.latitude)+"#Long#"+String((long)pos.longitude)+"#Speed#"+String(pos.knots)+"#Bearing#"+String(pos.course);

File dataFile = SD.open("datalog.txt", FILE\_WRITE);

if (dataFile) {

dataFile.println(output);

Serial.println(output);

dataFile.close();

// dataString outputs to the serial port

}

// If the file cannot be opened, the serial port outputs an error message

else {

Serial.println("error opening datalog");

}

}

}

void writeHeader(){

String header = "GPS data "+String(pos.date)+"\_"+String(pos.UTC)+".txt";

File dataFile = SD.open("datalog.txt", FILE\_WRITE);

dataFile.close();

if(SD.exists("datalog.txt")){

Serial.write("File Does exists\n");

dataFile = SD.open("datalog.txt", FILE\_WRITE);

}else{

Serial.write("File Does NOT exist\n");

}

if(dataFile){

dataFile.println(header);

}else{

Serial.println("Error opening datalog for header");

}

dataFile.close();

//Throttle Header

header = "Throttle\_data "+String(pos.date)+"\_"+String(pos.UTC)+".txt";

dataFile = SD.open("throttle.txt", FILE\_WRITE);

dataFile.close();

if(SD.exists("throttle.txt")){

Serial.write("File Does exists\n");

dataFile = SD.open("throttle.txt", FILE\_WRITE);

}else{

Serial.write("File Does NOT exist\n");

}

if(dataFile){

dataFile.println(header);

}else{

Serial.println("Error opening datalog for throttle header");

}

dataFile.close();

//Nav Header

header = "Nav\_data "+String(pos.date)+"\_"+String(pos.UTC)+".txt";

dataFile = SD.open("nav.txt", FILE\_WRITE);

dataFile.close();

if(SD.exists("nav.txt")){

Serial.write("File Does exists\n");

dataFile = SD.open("nav.txt", FILE\_WRITE);

}else{

Serial.write("File Does NOT exist\n");

}

if(dataFile){

dataFile.println(header);

}else{

Serial.println("Error opening datalog for nav header");

}

dataFile.close();

}

void writeThrottle(){

String throttleState = "<LEFT <DC "+String(leftSide.duty\_Cycle)+" /DC><Input "+String(leftSide.analog)+" /Input><us "+String(RC\*leftSide.duty\_Cycle) + " /us>/LEFT>\t";

throttleState += "<RIGHT <DC "+String(rightSide.duty\_Cycle)+" /DC><Input "+String(rightSide.analog)+" /Input><us "+String(RC\*rightSide.duty\_Cycle) + " /us>/RIGHT>";

File dataFile = SD.open("throttle.txt", FILE\_WRITE);

if(dataFile){

dataFile.println(throttleState);

}else{

Serial.println("Error opening throttle file.");

}

dataFile.close();

}

void writeNavigation(double dist, double bearing\_error, double target\_bearing){

String navigationErrors ="Target#Lat#"+String(points[targetIndex].latitudeDecimalDegrees,8)+"#Long#"+String(points[targetIndex].longitudeDecimalDegrees,8)+"#Bearing#"+String(target\_bearing,6)+

"#Current:#Lat#"+String(pos.latitudeDecimalDegrees,8)+"#Long#"+String(pos.longitudeDecimalDegrees,8)+"#Bearing#"+String(pos.course,6)+

"#Errors:#Dist#"+String(dist,6)+"#Bearing\_Error#"+String(bearing\_error,6)+

"#Motors:#Throttle%#"+String(throttle\_percentage)+"#Steering%#"+String(steering\_percentage)+"#Left\_DC#"+String(leftSide.duty\_Cycle,5)+"#Right\_DC#"+String(rightSide.duty\_Cycle,5)+"\n\n\n";

File dataFile = SD.open("nav.txt",FILE\_WRITE);

if(dataFile){

dataFile.println(navigationErrors);

}else{

Serial.println("Error opening nav error file");

}

dataFile.close();

}