/\* POWER-ASSIST SYSTEM - MK VI SOFTWARE \*\*\*\*\*\*\*\*\*\*\*\*\*\* ARDUINO UNO (ATMEGA328) APRIL 2021 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* BY: Project Manager: Steve Alvey 403-870-7210

\* SW developer: Steve Alvey 403-870-7210

\*

\* VERSION: 1.0RC POWER-ASSIST MK VI

\* Compatible with Mk VI REVC PCB (2021)

\*

\* PURPOSE: The POWER-ASSIST System provides control of HELM and WINCH 12V DC servo motors, to

\* control sails and steering system of small sailboats. Sailors with limited

\* hand function may control the Autohelm System via a wheelchair style joystick or Sip & Puff interface.

\* For safety, an (optional) RC Remote joystick is supported. When in use, commands received from the RC Remote Control

\* are given priority over operator joysitck or sip and puff commands. Once ANY RC input command is received,

\* Joystick and sip and puff commands are ignored for ONE SECOND. After ONE SECOND, joystick and sip and puff commands

\* will be handled.

\* See www.accessiblesailingtechnology.com for pictures and Power Assist System Operator Manual.

\*

\* LAST CHANGED: 4/06/21 S ALVEY - initial PRODUCTION BUILD tested and certified

\* Outstanding issues: NONE

\*

\* NOTES:

\* 1. N.O. pushbutton is connected to pin 1, to be used for future programming of MODES (programming LED present)

\* 2. pin 28 has tri-state input jumper, for use as hardware MODE input

\* 3. I2C interface may be added to pins A4 and A5 in future

\*

\*

\* GENERAL DESCRIPTION OF S/W:

\*

\* This software performs five functions continuously, four in the MAIN loop and the RC inputs via Interupt Service Routines (ISR):

\*

\* readbattery(); Battery voltage is polled continuously. When less than 12.V, Battery Low LED is flashed.

\* When less than 11.5V, Battery Low LED is ON and buzzer is ON as a warning to the Operator.

\* If battery voltage remains below 11.5V for more than ONE MINUTE, Helm and Winch outputs are disabled. Battery Low LED and buzzer remain ON.

\*

\* checkInputs(); The Arduino receives inputs from one of THREE SOURCES: A joystick on the Module; Sip & Puff sensors; or a remote control RC Joystick

\* Input commands are polled continuously and executed (one at a time), in priority: RC joystick commands are PRIORITY ONE and

\* will override conflicting joystick or sipp & puff commands received within ONE SECOND.

\* When a valid input command is received, that command must be released before any NEW command will be accepted.

\* Although the sip&puff and RC joystick inputs are analog (proportional), all inputs are sensed and decoded as BINARY,

\* once they exceed a "deadband" threshold. The deadband varies for each INPUT to establish an ergonomic sensitivity for that input device.

\* (i.e. joystick sensitivity is NOMINAL; sip & puff sensitivity is INCREASED; RC joystick sensitivity is DECREASED).

\*

\* handleInputCommand(); When an input command is received, it is decoded (joystick position) and then passed to be handled by handleInputCommand routine.

\* The HELM or WINCH actuator is activated in the appropriate direction for the period the INPUT is present. Only ONE INPUT will be handled at a time.

\* The HELM output is a proportional PWM drive for brushed DC motor, generating PWM (ANALOG) and PWM DIRECTION (DIGITAL) outputs.

\* Since all inputs are decoded as BINARY, the use of PWM proportional output is limited to the implementation of a "soft start" for the DC motor (PWM 'ramp').

\* The WINCH output actuates a DC Relay, located remotely. Two active HIGH signals: WINCHIN and WINCHOUT are generated on DIGITAL outputs, and

\* then inverted by a ULN2003 driver to actuate the DC Relay.

\*

\* heartbeat\_LED(); The heartbear LED 'blinks' at 1 Hz, indicating the Arduino is running. When an INPUT is present, the buzzer is activated at a 1 Hz rate,

\* providing audio feedback to the Operator. The buzzer is also used to warn the Operator that the battery is LOW.

\*

\* RC1/2\_Input; A remote control RC Joystick is an (optional) 'plug-n-play' device. When installed on the Mk VI PCB, the RC Receiver will generate two independent RC channels

\* (HELM = RC Receiver CH1, WINCH = RC Receiver CH3), each channel producing an PWM signal (RC standard: 1 - 2ms pulse at 50 Hz rate; 1.5ms nominal).

\* Interupt Service Routines timeRC1Pulse and timeRC2Pulse provide critical timing of the PWM pulse duration by synchronizing

\* with the rising and falling edges of each PWM pulse. These values are passed to the checkInputs routine for decoding.

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

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

/ define ATMega328 pins

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

#define interrupts()

#define USB\_TXD 0 // O PD0

#define PROGRAM 0 // O PD0

#define USB\_RXD 1 // I PD1

#define RC1\_INPUT 2 // I PB0

#define RC2\_INPUT 3 // I PB1

#define BUZZER 4 // O PD4

#define WINDLASS\_LMP 5 // O PD5

#define WINCH\_OUT 6 // O PD6

#define WINCH\_IN 7 // O PD7

#define PWM1\_DIRECTION 8 // O PD4

#define PWM1 9 // O PD3

#define BAT\_LOW\_LED 10 // O PB2

#define WINCH\_LED 11 // O PB3

#define HELM\_LED 12 // O PB4

#define HEARTBEAT\_LED 13 // O PB5

#define JOYSTICK\_HELM A0 // I PC0

#define JOYSTICK\_WIND A1 // I PC1

#define SIPPUFF\_HELM A2 // I PC2

#define SIPPUFF\_WIND A3 // I PC3

#define BATTERY A4 // I PC4

#define MODE A5 // I PC4

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

/ C++ function declarations

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

void readbattery(void);

void allOff(void);

byte checkInputs(void);

void handleInputCommand(void);

void heartbeat\_LED(void);

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

/ constants/variables for heartbeat timer routine

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

int ledState = LOW; // 'state' variable used to set the LED

bool beep = false; // boolean 'state' variable to activate BUZZER

unsigned long currentMicros = 0; // read the current time...

unsigned long previousMillis = 0; // store last time LED 'state' was changed

const long blink\_interval = 500; // set LED 'blink' interval (milliseconds)

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

/ constants/variables for joystick routine

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

byte JOY\_POS = 5; // joystick position: joy CENTER (no input) = 5; joy RIGHT = 4; joy LEFT = 3; joy UP = 2; joy DOWN = 1

int joy\_LR = 0; // variable for joystick helm value

int snp\_LR = 0; // variable for sip and puff helm value

int joy\_UD = 0; // variable for joystick winch value

int snp\_UD = 0; // variable for sip and puff winch value

const int J\_REF = 512; // set the centerpoint of joystick ADC

const int deadband = 100; // set joystick ADC deadband (512 +/- 100)

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

/ constants/variables for battery meter routine

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

int battery\_V = 0; // variable for battery voltage

byte beepState = LOW; // 'state' variable used to set the BUZZER

byte deadBatteryState = LOW; // 'state' variable used for dead battery voltage (less than 11.5V)

const int batterylow = 770; // set battery low constant @ 12.2V

const int batterydead = 725; // set battery dead constant @ 11.5V

unsigned long Battery\_State\_timer = 0; // timer used to measure duration of "dead battery" state

unsigned long Dead\_Battery\_detected = 0;// record the time when "dead battery" state (< 11.5V) is first detected; RESET this time whenever battery voltage recovers to ABOVE "dead battery" state (< 11.5V)

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

/ constants/variables for RC input routine

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

int RC1\_Pin = 0; // variable for reading RC1 pin

int RC2\_Pin = 0; // variable for reading RC2 pin

int RC1\_Value = 512; // variable for RC joystick winch value (512 = neutral)

int RC2\_Value = 512; // variable for RC sip and puff winch value (512 = neutral)

unsigned long RC\_NEUTRAL = 1500; // RC reference value, equivalent to 1.5msec pulsewidth

const long max\_pulse\_length = 2000; // longest valid pulse length is 2.0 msec

const long min\_pulse\_length = 500; // shortest valid pulse length is 500 millisec

bool startflag\_RC1\_Pulse = true;

bool startflag\_RC2\_Pulse = true;

byte bad\_RC1\_pulse = 0; // declared, but not used 040321

byte bad\_RC2\_pulse = 0; // declared, but not used 040321

unsigned long startofPulse = 0; // record the time of the 'start' of an RC pulse

unsigned long RC\_priority\_interval = 1000; // RC Commands are given PRIORITY. Once a command is received, joystick and s&p commands are

disabled for RC\_priority\_interval (ONE second = 1,000 milliseconds)

unsigned long RC\_Command\_received = 0; // initialized (0) each time RC command is received

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

/ constants/variables for PWM output generation

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

byte pwm\_value = 0; // value from 0 - 255 for PWM speed

byte pwm\_dir = 0; // value 1 or 0 for direction

const byte ramp\_inc = 2; // set joystick pwm ramp increment to implement 'soft start' of HELM actuator

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

void setup() //execute setup ONCE at start up, to initialize system states

{

pinMode(PROGRAM, INPUT); // button switch for programming/debugging

pinMode(JOYSTICK\_HELM, INPUT); // joystick helm

pinMode(JOYSTICK\_WIND, INPUT); // joystick windlass

pinMode(RC1\_INPUT, INPUT); // joystick helm

pinMode(RC2\_INPUT, INPUT); // joystick windlass

pinMode(SIPPUFF\_HELM, INPUT); // joystick helm

pinMode(SIPPUFF\_WIND, INPUT); // joystick windlass

pinMode(BATTERY, INPUT); // Battery voltage sense

pinMode(MODE, INPUT); // mode sense (tri-state)

pinMode(HEARTBEAT\_LED, OUTPUT); // Heartbeat LED

pinMode(BAT\_LOW\_LED, OUTPUT); // Bat Low LED

pinMode(WINCH\_LED, OUTPUT); // Winch LED

pinMode(HELM\_LED, OUTPUT); // Helm LED

pinMode(WINDLASS\_LMP, OUTPUT); // Remote LED on Windlass

pinMode(BUZZER, OUTPUT); // Piezo Buzzer

pinMode(PWM1, OUTPUT); // PWM1 helm

pinMode(PWM1\_DIRECTION, OUTPUT); // PWM1\_direction helm

Serial.begin(115200);

while(!Serial); // wait until SERIAL link is established....

setPwmFrequency(PWM1, 8); // increase Arduino PWM frequency to 3.9 KHz

attachInterrupt(digitalPinToInterrupt(RC1\_INPUT),timeRC1pulse,CHANGE); // attach interrupt handler for RC1\_INPUT. ISR called on both RISING and FALLING edge of PWM pulse

attachInterrupt(digitalPinToInterrupt(RC2\_INPUT),timeRC2pulse,CHANGE); // attach interrupt handler for RC2\_INPUT. ISR called on both RISING and FALLING edge of PWM pulse

allOff(); // initialize all outputs to OFF

} // END of setup()

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

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

\* CHANGE ARDUINO PWM FREQUENCY \* Allows Arduino PWM frequency to be raised from 488 Hz to 3.9 KHz for higher motor efficiency

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

void setPwmFrequency(int pin, int divisor)

{

byte mode;

if(pin == 5 || pin == 6 || pin == 9 || pin == 10) {

switch(divisor) {

case 1: mode = 0x01; break;

case 8: mode = 0x02; break;

case 64: mode = 0x03; break;

case 256: mode = 0x04; break;

case 1024: mode = 0x05; break;

default: return;

}

if(pin == 5 || pin == 6) {

TCCR0B = TCCR0B & 0b11111000 | mode;

} else {

TCCR1B = TCCR1B & 0b11111000 | mode;

}

} else if(pin == 3 || pin == 11) {

switch(divisor) {

case 1: mode = 0x01; break;

case 8: mode = 0x02; break;

case 32: mode = 0x03; break;

case 64: mode = 0x04; break;

case 128: mode = 0x05; break;

case 256: mode = 0x06; break;

case 1024: mode = 0x07; break;

default: return;

}

TCCR2B = TCCR2B & 0b11111000 | mode;

}

}

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

\* turn ALL off (MULTIPLE CALLS) \*

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

void allOff()

{

JOY\_POS = 5; // turn helm drive OFF

pwm\_value = 0; // turn helm drive OFF

RC1\_Value = J\_REF;

RC2\_Value = J\_REF;

digitalWrite(PWM1\_DIRECTION, pwm\_dir);

analogWrite(PWM1, pwm\_value);

digitalWrite(HELM\_LED, LOW);

digitalWrite(WINCH\_OUT, LOW); // TURN OFF WINCH AND WINCH LED

digitalWrite(WINCH\_IN, LOW);

digitalWrite(WINCH\_LED, LOW);

beep = false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* check battery voltage continuously

\* BATTERY METER ROUTINE \* when battery voltage drops below 12.2V, indicate with Battery Low LED flashing

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* when battery volatge drops below 11.5V, indicate with Battery Low LED ON, and buzzer ON

If battery voltage remians below 11.5V for more than ONE MINUTE, all outputs will be turned OFF to protect battery \*/

void readbattery()

{

unsigned long Battery\_State\_timer = millis(); // update Battery\_State\_timer

battery\_V = analogRead(BATTERY);

if (battery\_V < batterylow && battery\_V > batterydead) // if battery V greater than 11.5V but less than 12.2V: Battery Low LED flashing as "low battery" warning

{

digitalWrite(BAT\_LOW\_LED, ledState);

Dead\_Battery\_detected = millis(); //reset Dead\_Battery\_detected timer

}

else if (battery\_V < batterydead) // if battery V less than 11.5V (dead): Battery Low LED on full, BUZZER on as "dead battery" warning.

Then, if 'dead' voltage condition (< 11.5V) remains for ONE MINUTE, turn all outputs OFF (allOFF).

{

digitalWrite(BAT\_LOW\_LED, HIGH);

digitalWrite(BUZZER, HIGH);

if ((Battery\_State\_timer - Dead\_Battery\_detected) >= 60000) // if 'dead' voltage condition (< 11.5V) remains for ONE MINUTE (60,000 milliseconds), turn all outputs OFF (allOFF).

{

allOff();

}

}

else

{

digitalWrite(BAT\_LOW\_LED, LOW); // if battery V more than 12.2V: Battery OK

Dead\_Battery\_detected = millis(); //reset Dead\_Battery\_detected timer

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* check for joystick, RC, Sip & Puff input command

\* CHECK INPUT COMMANDS \* RC Joystick is offered as an optional SAFETY device, for Disabled Sailing Program Operator.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* if RC Joystick (option) is present and turned ON, these commands will be given PRIORITY over a sailor's joystick or sip & puff commands \*/

byte checkInputs()

{

unsigned long RC\_Priority\_timer = millis(); // RC Commands are given PRIORITY ONE. Once an RC command is received, joystick and s&p commands are disabled for ONE second

joy\_LR = analogRead(JOYSTICK\_HELM);

snp\_LR = analogRead(SIPPUFF\_HELM);

joy\_UD = (1024 - analogRead(JOYSTICK\_WIND)); // NOTE: reverse polarity of joystick up/down

snp\_UD = (analogRead(SIPPUFF\_WIND));

if (RC1\_Value > (J\_REF + (deadband\*2))) // RC joystick deflection (PRIOIRTY ONE) greater than deadband = HELM PORT COMMAND

{

JOY\_POS = 3;

RC\_Command\_received = millis();

return;

}

else if (RC1\_Value < (J\_REF - (deadband\*2))) // RC joystick deflection (PRIOIRTY ONE) less than deadband = HELM STARBOARD COMMAND

{

JOY\_POS = 4;

RC\_Command\_received = millis();

return;

}

else if (RC2\_Value > (J\_REF + (deadband\*3))) // RC joystick deflection (PRIOIRTY ONE) greater than deadband = WINCH OUT COMMAND

{

JOY\_POS = 1;

RC\_Command\_received = millis();

return;

}

else if (RC2\_Value < (J\_REF - (deadband\*3))) // RC joystick deflection (PRIOIRTY ONE) less than (neg) deadband = WINCH IN COMMAND

{

JOY\_POS = 2;

RC\_Command\_received = millis();

return;

}

if ((RC\_Priority\_timer - RC\_Command\_received) >= RC\_priority\_interval) // RC Commands are given PRIORITY ONE. Once an RC command is received,

joystick and s&p commnads are disabled for RC\_priority\_interval (ONE second)

{

if ((joy\_LR > (J\_REF + deadband)) || (snp\_LR > (J\_REF + (deadband/2)))) // joystick deflection (or sip/puff) (PRIORITY TWO) greater than deadband = HELM PORT COMMAND

{

JOY\_POS = 3;

return;

}

else if ((joy\_LR < (J\_REF - deadband)) || (snp\_LR < (J\_REF - (deadband/2)))) // joystick deflection (or sip/puff) (PRIORITY TWO) less than deadband = HELM STARBOARD COMMAND

{

JOY\_POS = 4;

return;

}

else if ((joy\_UD > (J\_REF + deadband)) || (snp\_UD > (J\_REF + (deadband/2)))) // joystick deflection (or sip/puff) (PRIORITY TWO) greater than deadband = WINCH OUT COMMAND

{

JOY\_POS = 1;

return;

}

else if ((joy\_UD < (J\_REF - deadband)) || (snp\_UD < (J\_REF - (deadband/2)))) // joystick deflection (or sip/puff) (PRIORITY TWO) less than (neg) deadband = WINCH IN COMMAND

{

JOY\_POS = 2;

return;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* when input command received, execute requested OUTPUT

\* HANDLE INPUT COMMANDS \* for safety, only one OUTPUT command can be executed at one time

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

void handleInputCommand()

{

//Serial.println(JOY\_POS);

switch(JOY\_POS)

{

case 1:

{

digitalWrite(WINCH\_OUT, HIGH); // if Joystick position = 1, actuate WINCH OUT

digitalWrite(WINCH\_LED, HIGH);

beep = true;

JOY\_POS = 5; // command complete; RESET joystick position flag

break;

}

case 2:

{

digitalWrite(WINCH\_IN, HIGH); // if Joystick position = 2, actuate WINCH IN

digitalWrite(WINCH\_LED, HIGH);

beep = true;

JOY\_POS = 5; // command complete; RESET joystick position flag

break;

}

case 3:

{

pwm\_dir = 1; // if Joystick position = 3, actuate HELM to port

if (pwm\_value < 250) // pwm output value is 'ramped' from 0 - 255 to implement 'soft start' for HELM output

{

pwm\_value = (pwm\_value + ramp\_inc); // increment 'pwm ramp' value

beep = true;

delay(1);

}

else

{

pwm\_value = 255; // go FULL SPEED

beep = true;

}

digitalWrite(PWM1\_DIRECTION, pwm\_dir);

analogWrite(PWM1, pwm\_value);

digitalWrite(HELM\_LED, HIGH);

JOY\_POS = 5; // command complete; RESET joystick position flag

break;

}

case 4:

{

pwm\_dir = 0; // if Joystick position = 4, actuate HELM to starboard

if (pwm\_value < 250) // pwm output value is 'ramped' from 0 - 255 to implement 'soft start' for HELM output

{

pwm\_value = (pwm\_value + ramp\_inc); // increment 'pwm ramp' value

beep = true;

delay(1);

}

else

{

pwm\_value = 255; // go FULL SPEED

beep = true;

}

digitalWrite(PWM1\_DIRECTION, pwm\_dir);

analogWrite(PWM1, pwm\_value);

digitalWrite(HELM\_LED, HIGH);

JOY\_POS = 5; // command complete; RESET joystick position flag

break;

}

case 5: // if Joystick position = 5; no INPUT is present

{

allOff(); // turn all outputs OFF

break;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* check to see if it's time to blink the LED: that is, if the difference

\* HEARTBEAT LED ROUTINE \* between the current time and last time you blinked the LED is greater than

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* the 'blink' period (blink\_interval) \*/

void heartbeat\_LED()

{

unsigned long currentMillis = millis(); // read the current time...

if (currentMillis - previousMillis >= blink\_interval) // once 'blink\_interval' time reached....

{

previousMillis = currentMillis; // ....save the last time you blinked the LED and then....

if (ledState == LOW)

{

ledState = HIGH; // ....toggle state

if (beep == true) beepState = true;

else beepState = false;

}

else

{

ledState = LOW;

beepState = false;

}

digitalWrite(HEARTBEAT\_LED, ledState); // write the HEARTBEAT LED with 'ledState'

digitalWrite(WINDLASS\_LMP, ledState);

digitalWrite(BUZZER, beepState); // write the BUZZER with 'beepState'

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* // synchronize with the rising edge of RC1 pulse, then measure pulse width

\* RC1 INPUT INTERUPT SERVICE ROUTINE \*

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

void timeRC1pulse()

{

noInterrupts();

if ((digitalRead (RC1\_INPUT) == HIGH) && (startflag\_RC1\_Pulse == true)) // if interupt is RISING edge of new PWM pulse, record time of start of PWM pulse

{

startofPulse = micros();

startflag\_RC1\_Pulse = false;

interrupts();

return;

}

else // if interupt is FALLING edge of PWM pulse, validate and then measure width of PWM pulse

{

currentMicros = micros();

RC1\_Value = (currentMicros - startofPulse);

if ((RC1\_Value > max\_pulse\_length)||(RC1\_Value < min\_pulse\_length))

{

bad\_RC1\_pulse = 1; // validate pulse width > min\_pulse\_length

RC1\_Value = 512;

}

else

{

RC1\_Value = constrain(RC1\_Value, 1000, 2000); // CONSTRAIN pulse width to VALID range (1000 - 2000 microsec)

RC1\_Value = map(RC1\_Value, 1000, 2000, 0, 1024); // finally, SHIFT final RC value to VALID input range (comparable to joystick and sip and puff input range of 0 - 1024)

}

startflag\_RC1\_Pulse = true; // prepare to receive next PWM pulse

interrupts();

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* // synchronize with the rising edge of RC2 pulse, then measure pulse width

\* RC2 INPUT INTERUPT SERVICE ROUTINE \*

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

void timeRC2pulse()

{

noInterrupts();

if ((digitalRead (RC2\_INPUT) == HIGH) && (startflag\_RC2\_Pulse == true)) // if interupt is RISING edge of new PWM pulse, recond time of start of PWM pulse

{

startofPulse = micros();

startflag\_RC2\_Pulse = false;

interrupts();

return;

}

else // if interupt is FALLING edge of PWM pulse, validate and then measure width of PWM pulse

{

currentMicros = micros();

RC2\_Value = (currentMicros - startofPulse);

if ((RC2\_Value > max\_pulse\_length)||(RC2\_Value < min\_pulse\_length))

{

bad\_RC2\_pulse = 1; // validate pulse width > min\_pulse\_length

RC2\_Value = 512;

}

else

{

RC2\_Value = constrain(RC2\_Value, 1000, 2000); // CONSTRAIN pulse width to VALID range (1000 - 2000 microsec)

RC2\_Value = map(RC2\_Value, 1000, 2000, 0, 1024); // finally, SHIFT final RC value to VALID input range (comparable to joystick and sip and puff input range of 0 - 1024)

}

startflag\_RC2\_Pulse = true; // prepare to receive next PWM pulse

interrupts();

}

}

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

\*

\* MAIN PROGRAM LOOP \* MAIN program loop; executes continuously

\*

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

void loop()

{

readbattery(); // battery voltage is polled continuously. When less than 12.2V, Battery Low LED is flashed. When less than 11.5V, Battery Low LED is ON and buzzer is ON; HELM and WINCH are disabled.

checkInputs(); // Input commands are polled continuously; for safety, RC commands are PRIORITY ONE and override conflicting joystick and sipp & puff commands for ONE second

handleInputCommand(); // when present, input commands are handled by actuating HELM or WINCH as required

heartbeat\_LED(); // heartbeat LED is blinked at 1 Hz; buzzer is activated when input command is present

}