Adaptive RC Sailer Project Design Notes – Phase Three

Demonstration Model/Prototype (Pulse Position Modulation With Voice-Control Analog to Digital Inputs)

Please refer to the *Phase Two Adaptive RC Sailer Interface Design Notes* for a review of the Pulse Position Modulation and previous binary input design approach and methodologies.

The Phase Three prototype uses an Arduino UNO microcontroller to control a FlySky i6X RC transmitter via its Pulse Position Modulation (PPM) Trainer Port. The control signal inputs are derived from serial communication with the Arduino via a (Geeetech) voice-control module. Input signal functions are the same as in Phase Two, i.e. the On & Off impulses from Phase Two are essentially digital On & Off impulses from the voice-control module and interpreted/handled through the Arduino source code. Input signals into the voice module include the commands: "Left", "Right", "In", "Out", and "Center". "Left" and "Right" move the rudder accordingly, "In and "Out" move the sails accordingly, and "Center" moves the rudder back to a zero-turn state.

The PPM utility remains the same as in Phase Two. No modifications are required for either the RC radio system or sailboat systems and no internet connection is required.

Again, special thanks to Gabriel Staples https://www.electricrcaircraftguy.com/ for his eRCaGuy_PPM_Writer library utility.

Below is an explanation of the major components, notes, and warnings related to the Phase Three Arduino Source Code:

All channels and information regarding the PPM creation/modification within the code remain the same as in Phase 2. For reference, please see *Adaptive RC Sailer Project Design Notes - Phase Two* or see below for a copy of this information.

Sail Control = Channel 3 : Rudder Control = Channel 1

In order to facilitate smooth operation and account for skipper input lag, system delays were implemented for both the sails and rudder.

Nominal Channel Width = 1500 msec +/- 300 msec

getPPMPolarity

PPM_WRITER_NORMAL has a HIGH base-line signal, with channelSpace pulses that are LOW

Sails = Channel 3

Number of Click from Sails Full In to Sails Full Out = 12

Sails_Step = 500 msec / 12 = 50 msec

Sails_In = -50 msec

Sails_Out = +50 msec

Present Position = pulsewidth_Sails

New Position = Present Position +/- Sails_Step

Restrain Position 1200 to 1800 msec

PPMWriter Code : PPMWriter.setChannelVal(CH3, pulsewidth_Sails * 2);

Rudders = Channel 1

Number of Click from Rudder Full Left to Rudder Full Right = 16

center = 1;

This variable changes to enable the rudder-centering algorithm to begin.

Rudder_Step = 600 msec / 16 = 37.5 = 36 msec

Rudder_Left = -36 msec

Rudder_Right = +36 msec

Present Position = pulsewidth_Rudder

New Position = Present Position +/- Rudder_Step

Restrain Position 1200 to 1800 msec

PPMWriter Code : PPMWriter.setChannelVal(CH4, pulsewidth_Rudder * 2);

getPPMPolarity

-PPM_WRITER_NORMAL has a HIGH base-line signal, with channelSpace pulses that are LOW

FlySky i6X Training Mode Switchology (Software Setup)

Training Mode = ON

Student Mode = OFF

Switch "D" = Engaged (Down Position)

Phase 3 Modifications, Changes, and Important Notes:

CAUTION: As in Phase Two, the Arduino output on pin D9 is nominally 5 VDC. The FlySky i6X is basically a 3.3 VDC system. A resistive voltage divider network was then incorporated where R1 = 332 Ω and R2 = 649 Ω. (These values were changed from Phase Two but either solutions will work).

Vout = R2/(R1 + R2) * Vin = 649/(332 + 649) * 5 = 3.307 VDC

TRAINING THE VOICE MODULE: The voice module used to communicate serially with the Arduino software in Phase Three is the *Geeetech Arduino Voice Recognition Module*.

Information on this module is available here:
https://www.geeetech.com/wiki/index.php/Arduino_Voice_Recognition_Module

And more detailed training information is available here:

https://www.geeetech.com/wiki/images/6/69/Voice_Recognize_manual.pdf

When the term "train" or "training" is used, it refers to setting up the voice module to recognize your voice commands. This can be done in two ways:

1. Using the Arduino
   During training:
   -Connect the receiving line (RX) to receiving on Geeetech module (RX)
   -Connect the transmission line (TX) to transmission on Geeetech module (TX)
   During Use:
   -Connect the receiving line (RX) to the transmit on Geeetech module (TX)
   -Connect the transmission line (TX) to the receiving on Geeetech module (RX)

2. Using an RS 232 TTL module
   During training:
   -Connect the receiving line (RX) to the transmission on Geeetech module (TX)
   -Connect the transmission line (TX) to the receiving on Geeetech module (RX)

Both options use the free-to-download AccessPort137 Software:

Available Here: https://accessport.en.lo4d.com/windows

Note: Please delete the previous user's voice commands *prior to training a new voice* by using the hex commands: xAA 01, xAA 02, or xAA 03 to delete groups 1-3, respectively.

Note: Please refer to the voice-control module training guide for step-by-step instructions on how to use AccessPort137 to train the Geeetech voice module.

IMPORTANT PHASE THREE SOFTWARE NOTES:

The Phase Three software source code only works with the Geeetech brand/style of voice module.

The Phase Three software will only output correct movements if the voice is trained in the following order:

1. "Left"
2. "Right"
3. "In"
4. "Out"
5. "Center"

Note: With the software currently used in Phase Three, the actual words used can be changed to suit the user, but the word used to move the rudder left, for example, must *always* be the first word recorded, the word to move it right must be second, and so on for the remaining commands.

The reason for this strict order of commands when training is the code itself. The checks used to change the correct input variable are in the Loop() method of the code with the order of:

```
while(Serial.available()){

com = Serial.read();

switch(com){

//left movement command first
case 0x11:

Serial.println("Left");
RudderLeft = 0;

break;

//right movement command second
case 0x12:

Serial.println("Right");
RudderRight = 0;

break;

//sails in command third
case 0x13:

Serial.println("Sails in");
SailsIn = 0;

break;

//sails out command fourth
case 0x14:

Serial.println("Sails out");
SailsOut = 0;

break;
```

```
//center command fifth
case 0x15:

Serial.println("Center");
center = 0;

break;

}
```

Note: Because the software relies on serial communication, if the Serial is unavailable or flooded with noise the above checks will fail and the device/prototype will not function correctly. If this occurs, verify that the RX and TX lines are connected properly according to the usage instructions above.

IMPORTANT ADDITIONS MADE TO THE PHASE TWO SETUP() METHOD:

The serial communication is started in the Setup() method.

CAUTION: Timing is important for the Geeetech module. The three second delay between header bytes is necessary and should only be made LONGER if desired, not shorter.

> -The Geeetech module should begin to slow flash after this method runs. If it does not, unplug the Geeetech module and plug it back in again.

```
Serial.begin(9600);
//wait 3 seconds before sending header bytes to allow Geeetech time to respond
delay(3000);
Serial.write(0xAA); //header byte
Serial.write(0x37); //key byte of x37 means switch to compact mode

//send same header again to avoid potential errors in missed bytes with the Geeetech
Serial.println("Header xAA37 sent");
Serial.write(0xAA); //header byte
Serial.write(0x37); //key byte of x37 means switch to compact mode
Serial.println("Header xAA37 sent");

//Note: If using group 1: use 0x21, if using group 2: use 0x22, if using group 3: use 0x23
//This imports the commands. If this header is not sent, the Geeetech will not enter into waiting
//mode and the device will not function properly.
delay(3000);
Serial.write(0xAA); //header byte
Serial.write(0x21); //key byte of 21 means import group 1 commands and be ready for voice
```

RUDDER-CENTERING ALGORITHM: The Phase Three software includes a new algorithm designed to center the rudder back to a zero-turn state once the word “center” is given. It is included below with comments on its major parts/variables.

Note: The algorithm and comments explaining its major parts appears below.

```
   if(center == 0){
//if the rudder is facing right (greater than 0), decrease by 1 every 20 ms and adjust the PPM value accordingly
  if(RudderCount > 0){
    while(RudderCount > 0){
      RudderCount = RudderCount - 1;
          //keep track of the rudder count and its position and decrement it each time
      Serial.print("RudderCount: ");
      Serial.println(RudderCount);
      PPMWriter.setChannelVal(CH1, (RudderPulseWidth + RudderCount * RudderPulseWidthInc) * 2);
          //the delay between increment shifts can be modified to speed up or slow down the rate at which the rudder moves back to the center position
      delay(20);
    }
  }

//if the rudder is facing left (less than 0), decrease by 1 every 20 ms and adjust the PPM value accordingly
  if(RudderCount < 0){
    while(RudderCount < 0){
      RudderCount = RudderCount + 1;
          //keep track of the rudder count and position
      PPMWriter.setChannelVal(CH1, (RudderPulseWidth + RudderCount * RudderPulseWidthInc) * 2);
      Serial.print("RudderCount: ");
      Serial.println(RudderCount);
          //this delay can be changed to speed up or slow down the rudder’s return rate
      delay(20);
    }
  }
   }
```

Note: Because of the rudder-centering algorithm and the way the Phase Three software checks for and decides that a command has been said, the following addition was made to the end of the Loop() method after the switch-case checks are made with every loop.

```
//go through the rudder and sail methods to redefine their values
//set each variable back to 1 to avoid constantly affecting the rudder or sail PPM value
rudder_control();
RudderLeft = 1;
RudderRight = 1;
center = 1;
Serial.println("Rudder values reset to 1");

sails_control();
SailsOut = 1;
SailsIn = 1;
Serial.println("Sail values reset to 1");
```

CAUTION: This portion is important as without resetting the key variables of RudderLeft, RudderRight, center, SailsOut, and SailsIn back to a “false” state of 1, the program would get stuck on the last command given and would never give the option to produce a different command.