Code to Drive Pan-Tilt Head

```
#include <Servo.h>
Servo YawServo, PitchServo;

#include <LiquidCrystal.h>
LiquidCrystal LcdDriver(11, 9, 5, 6, 7, 8);
//Set up clock.
#define LcdDisplay
#include "ClockBasics.h"
unsigned long ClockTimer;

// Servo Parameters
int YawValue = 90;
int PitchValue = 90;
unsigned long ScanTimer;
#define ScanStepInterval  90

#include "EncoderRead.h"
// State showing which axis if currently being changed.
enum EncoderControlStates { Yaw, Pitch, Scan };
EncoderControlStates EncoderControl = Yaw;

// Variable used to track encoder movements.
volatile int CurrentEncoder = EncoderValue;
#define CountsPerStep 4

// Function called in loop to check for encoder movement
// and then update the active axis.
int UpdateFromEncoder()
{
	int ReturnValue; // Default return
	ReturnValue = 0;
	// Check for CCW turn
	if (CurrentEncoder - EncoderValue < -CountsPerStep)
	{
		// Change Yaw or Pitch based on control state
		switch (EncoderControl)
		{
		case Yaw:
			YawValue++;
			break;
		case Pitch:
			PitchValue++;
			break;
		case Scan:
			break;
		}
		// Update CurrentEncoder.
		CurrentEncoder = EncoderValue;
		//
		ReturnValue = 1;
	}// End of CCW check
	// Check for CW turn
	if (CurrentEncoder - EncoderValue  > CountsPerStep)
	{
		// Change Yaw or Pitch based on control state
		switch (EncoderControl)
		{
		case Yaw:
			YawValue--;
			break;
		case Pitch:
			PitchValue--;
			break;
```

```
                case Scan:
                        break;
                }
                // Update CurrentEncoder.
                CurrentEncoder = EncoderValue;
                ReturnValue = 2;
        } // End of CW check

        // Set limits on Pitch and Yaw
        if (YawValue > 180)
                YawValue = 180;
        else if (YawValue < 0)
                YawValue = 0;
        if (PitchValue > 180)
                PitchValue = 180;
        else if (PitchValue < 0)
                PitchValue = 0;

        return ReturnValue;

} // End of UpdateFromEncoder

#include "ButtonDebounce.h"

void UpdateLCD()
{
        LcdDriver.clear();// Reset display
        LcdDriver.setCursor(0, 0);
        LcdDriver.print("Y= "); // place yaw on first line
        LcdDriver.print(YawValue);
        LcdDriver.setCursor(0, 1);
        LcdDriver.print("P= ");  // place pitch on second
        LcdDriver.print(PitchValue);
        LcdDriver.setCursor(12, 0);
        LcdDriver.print("Scan");
        LcdDriver.setCursor(7, 1);
        LcdClock();
        // Place cursor on line that is currently being changed.
        switch (EncoderControl)
        {
        case Yaw:
                LcdDriver.setCursor(2, 0);
                break;
        case Pitch:
                LcdDriver.setCursor(2, 1);
                break;
        case Scan:
                LcdDriver.setCursor(11, 0);
        }
        // Be sure cursor is on and blinking.
        LcdDriver.cursor();
        LcdDriver.blink();
} // End of UpdateLCD
```

```
// put your setup code here, to run once:
void setup() {
	// Set up Encoder
	EncoderInitialize();
	// Set up button
	ButtonInitialize();
	// Set up servos.
	YawServo.attach(10);
	YawServo.write(90);
	YawValue = 90;
	PitchServo.attach(12);
	PitchServo.write(90);
	PitchValue = 90;

	// Set up software timers.
	ClockTimer = millis();
	ScanTimer = millis();

	// Set up lcd.
	LcdDriver.begin(16, 2);
	UpdateLCD();
	Serial.begin(9600);
}

// put your main code here, to run repeatedly:
void loop()
{
	int OtherUpdates = 0;
	// Check for button press
	if (1 == ButtonRead())
	{
		// then switch active axis.
		if (EncoderControl == Yaw)
		{
			EncoderControl = Pitch;
		}
		else if (EncoderControl == Pitch)
		{
			EncoderControl = Scan;
			YawValue = 150;
			PitchValue = 100;
		}
		else
		{
			EncoderControl = Yaw;
		} // End of EncoderControl if

		UpdateLCD(); // Update display to reflect change.
	} // End of Button if

	if (millis() - ScanTimer > ScanStepInterval)
	{
		if (EncoderControl == Scan)
		{
			PitchValue -= 2;
			if (PitchValue < 45)
			{
				PitchValue = 105;
				YawValue += 5;
				if (YawValue > 120)
					YawValue = 60;
			}
			OtherUpdates = 1;
		}
		ScanTimer += ScanStepInterval;
	}
```

```
	if (millis() - ClockTimer >= 1000)
	{
		ClockTimer += 1000;
		// if clock is running, update the clock
		if (clockState == CLOCK_RUNNING)
			UpdateClock();
		// Then send data out.
		OtherUpdates = 1;
		// update timer

	} // End of ClockTimer if

	// Check for incoming data
	if (Serial.available())
	{
		// Use character to set clock.
		SettingClock(Serial.read());

	} // End of Serial input handling

	// Check for update from encoder.
	if (UpdateFromEncoder() || OtherUpdates)
	{
		OtherUpdates = 0;
		// Write to servos
		YawServo.write(YawValue);
		PitchServo.write(PitchValue);
		UpdateLCD(); // update display reflecting change of axes.
	}

}
```

Support Code for reading Button

```
#ifndef ButtonDebounce_H
#define ButtonDebounce_H

// Set up pin and button state.
int ButtonPin = 4, buttonState = 0;
unsigned long buttonTimer;

// Initialization code, setting up pin.
void ButtonInitialize()
{
	pinMode(ButtonPin, INPUT);
} // End of ButtonInitialize

// Function called in loop to check for button release.
// Returns a 1 on the buttons release.
int ButtonRead()
{
	// Read in the buttons current value.
	int Press = digitalRead(ButtonPin);
	int ReturnValue = 0;
	switch (buttonState)
	{
	case 0: // if we are waiting for a press,
		if (Press == LOW)
		{
			// Once press occurs
			buttonTimer = millis();  // record time
			buttonState = 1;        // and move to next state
		}
		break;
	case 1: // button just went low
		if (Press == HIGH) // and now goes high
		{
			buttonState = 0;  // return to 0 state.
		}
		else // if still low
		{
			// and sufficient time has passed.
			if (millis() - buttonTimer >= 10)
			{
				buttonState = 2; // move on to state two
			}
		}
		break;
	case 2:
		if (Press == HIGH)
		{
			ReturnValue = 1; // Return 1 indicating release.
			buttonState = 0;
		} // End of high test.
		break;
	} // End of switch on buttonState

	return ReturnValue;

} // End of ButtonRead

#endif
```

Support Code for Reading Encoder

```
#ifndef EncoderRead_H
#define EncoderRead_H

// Variable for keeping track of encoder change.
volatile int EncoderValue = 0;
int EncAPin = 2, EncBPin = 3;
int EncA, EncB;

// Service Routine for Encoder channel A,
// active on channel A changing.
void PinA(void)
{
	// Check the two inputs and then if not equal
	if (digitalRead(EncBPin) != digitalRead(EncAPin))
	{
		EncoderValue++; // Increment the encoder.
	}
	else
	{
		EncoderValue--; // otherwise decrement
	}
} // End of PinA

// Service Routine for Encoder channel B,
// active on channel B changing.
void PinB(void)
{
	// Check the two inputs and then if equal
	if (digitalRead(EncBPin) == digitalRead(EncAPin))
	{
		EncoderValue++; // Increment the encoder.
	}
	else
	{
		EncoderValue--;// otherwise decrement
	}
} // End of PinB

// Setup interrupt services and pinModes for the Encoder lines.
void EncoderInitialize()
{
	attachInterrupt(0, PinA, CHANGE); // ISR's
	attachInterrupt(1, PinB, CHANGE);
	pinMode(EncAPin, INPUT); // Pin Modes.
	pinMode(EncBPin, INPUT);
}// End of EncoderInitialize

#endif
```

Support Code for Handling Clock.

```
#ifndef ClockBasics_H
#define ClockBasics_H
// Variable used as clock settings.
int Hours, Minutes, Seconds;
// This function is to be called every second
// to update the clock represented by the
// global variables Hours, Minutes, Seconds
void UpdateClock()
{
	// Check if Seconds not at wrap point.
	if (Seconds < 59) {
		Seconds++; // Move seconds ahead.
	}
	else {
		Seconds = 0; // Reset Seconds
		// and check Minutes for wrap.
		if (Minutes < 59) {
			Minutes++; // Move seconds ahead.
		}
		else {
			Minutes = 0; // Reset Minutes
			// check Hours for wrap
			if (Hours < 23)
			{
				Hours++;// Move Hours ahead.
			}
			else
			{
				Hours = 0;// Reset Hours
			}// End of Hours test.
		} // End of Minutes test
	} // End of Seconds test
} // end of UpdateClock()
void SendClock()
{
	// Check if leading zero needs to be sent
	if (Hours < 10)
	{
		Serial.print("0");
	}
	Serial.print(Hours); // Then send hours
	Serial.print(":"); // And separator
	// Check for leading zero on Minutes.
	if (Minutes < 10)
	{
		Serial.print("0");
	}
	Serial.print(Minutes); // Then send Minutes
	Serial.print(":"); // And separator
	// Check for leading zero needed for Seconds.
	if (Seconds < 10)
	{
		Serial.print("0");
	}
	Serial.println(Seconds); // Then send Seconds
	// with new line
} // End of SendClock()
```

```
#ifdef LcdDisplay
void LcdClock()
{
	// Reset display
	// Check if leading zero needs to be sent
	if (Hours < 10)
	{
		LcdDriver.print("0");
	}
	LcdDriver.print(Hours); // Then send hours
	LcdDriver.print(":"); // And separator
	// Check for leading zero on Minutes.
	if (Minutes < 10)
	{
		LcdDriver.print("0");
	}
	LcdDriver.print(Minutes); // Then send Minutes
	LcdDriver.print(":"); // And separator
	// Check for leading zero needed for Seconds.
	if (Seconds < 10)
	{
		LcdDriver.print("0");
	}
	LcdDriver.print(Seconds); // Then send Seconds
	// with new line
} // End of SendClock()

#endif

// States for setting clock.
enum ClockStates {
	CLOCK_RUNNING, CLOCK_SET_HOURS,
	CLOCK_SET_MINUTES, CLOCK_SET_SECONDS
};
ClockStates clockState = CLOCK_RUNNING;

// Function that processes incoming characters to set the clock.
void SettingClock(char Input)
{
	// interpret input based on state
	switch (clockState)
	{
	case CLOCK_RUNNING:
		if (Input == 'S') // Move to Setting clock.
		{
			clockState = CLOCK_SET_HOURS;
			Hours = 0;   // Reset clock values.
			Minutes = 0;
			Seconds = 0;
		}
		break;
	case CLOCK_SET_HOURS: // In process of setting clock hours.
		if (Input >= '0' && Input <= '9')
			Hours = 10 * (Hours % 10) + Input - '0';
		else if (Input == ':')
			clockState = CLOCK_SET_MINUTES;
		else if (Input == 'R')
			clockState = CLOCK_RUNNING;
		break;
```

```
	case CLOCK_SET_MINUTES: // In process of setting clock minutes.
		if (Input >= '0' && Input <= '9')
			Minutes = 10 * (Minutes % 10) + Input - '0';
		else if (Input == ':')
			clockState = CLOCK_SET_SECONDS;
		else if (Input == 'R')
			clockState = CLOCK_RUNNING;
		break;
	case CLOCK_SET_SECONDS: // In process of setting clock seconds.
		if (Input >= '0' && Input <= '9')
			Seconds = 10 * (Seconds % 10) + Input - '0';
		else if (Input == 'R')
			clockState = CLOCK_RUNNING;
		break;

	}// End of clock mode switch.

} // End of SettingClock

#endif
```