# Crayfish Stretch Receptor Stimulator

Report for Cornell University

ECE MEng design project


**By**

Zequn Huang

Ningning Ding

Jiachen Hu

**Project Advisor:**

Bruce Land

Bruce Johnson

**ABSTRACT**

This project aims to create a microcontroller control system to stimulate the crayfish tail stretch receptor. In area of biology, a crayfish is a commonly used animal model for understanding sensory function, and for our project crayfish tail stretch receptor is studied in introductory neurobiology labs. In order to automate the crayfish tail receptor stimulator, a servo motor driven by a microcontroller was used, and several control inputs were designed on the user interface board connected to the microcontroller for factor controls. At the end of the design project, the automated crayfish stretch receptor stimulator was able to pull the crayfish tail a known amount at a consistent speed. The design is able to let the user vary the pulling amount and the speeds, and even set up a paradigm of different pulls and speeds during the same epoch. With more steady and more accurate pulls than manual stimulator, such a stimulator makes things easier for learners to understand related sensory knowledge in crayfish tail receptor labs and for researchers to do study in crayfish tail receptor experiments.

## *EXECUTIVE SUMMARY*

Our project aims to design an automatic crayfish tail receptor stimulator. Such a device has already existed in biology labs, but it is a human manipulated device with which people can only pull the crayfish tail in a qualitative way. As such a lab is in need of being more accurate and more quantitative, an accurate automatic device is required more often to replace the old one.

Before stat designing there are some issues to be considered and carefully thought. In order to make a better automatic device the original one's pros and cons should be noticed. The old stimulator is made of steel, heavy enough to be manipulated so that the device is stable to use. However the old stimulator has very little control on the stretching speed, and it is hard to drag the tail same distance every time. Except these, the pulling speed is also quit slow and less consistent, which is not good for our sensitive nerve testing. So for the new one we need to consider the setting of drag positions and speed to make the device easier to conduct labs, and we should consider the stability of the dragging device. Apart from that we should also carefully consider how to implement those functions, since we use microcontrollers instead of mechanical controls to design such a device. Another issue to consider is the fact that the electrical servo may be quit noisy. Since the crayfish nerve testing signal is highly sensitive to electrical noise, this problem might be severe if the noise affects the nerve signal. We decide to use a low-noise servo to reduce generated noises.

We divided our design into three parts, hardware circuit part, software coding part and mechanical part. For the circuits, we use a microcontroller to act as the main control unit of the device. Apart from that we chose a low-noise servo motor to drag the crayfish tail so that less noise is generated during the lab, and we used potentiometer knob as input to adjust the position and the dragging speed. A LED light is used to indicate the power on/off, and a wire is used to

connect to the oscilloscope to notify the start of the drag for recording. For software coding part, we designed a finite state machine to control the manipulation of the whole device, including position setting, speed adjusting and drag control. For mechanical design, a box and a lab support are used to form the whole stimulator, with microcontroller circuits in the box and the servo motor on the metal support (for stability). A user-friendly control panel was designed to control the whole stimulator.

The device proved to be a better stimulator than the original one. It is able to drag the crayfish tail from different starting positions and to various ending positions, and the pulling speed can be set to six different kinds. It is able to the tail as stable as the original one because the servo is put on the heavy lab support. More importantly, the lab result indicates that there is almost no noise added by the device, and the result is very easy for researchers to analyze.

## *INTRODUCTION*

Crayfish stretch receptor is a part of crayfish nervous system and it is able to monitor the movement and the position of crayfish body. To better understand about it, crayfish stretch receptor labs are designed, in which dissected crayfish tails are stretched using manipulators to make receptor generate voltage signals. In labs various pulling ways are used to monitor the operation of crayfish stretch receptor comprehensively.

Although such labs can be done by manually pulling the crayfish tail, the speeds of stretches are hard to control by doing so, and personal errors might be generated in terms of pulling distance. All such inconveniences call for a better receptor stimulator, and that is what this project aims to – an automatic stimulator that pulls more steadily and more accurately.

This project is a three-member teamwork that ended up with an automated crayfish tail receptor stimulator. Based on the analyzed issues in the proposal and careful design, the

stimulator was realized successfully and all functionalities were implemented. The project report is written in the following parts: the background section describes some context knowledge of the crayfish stretch receptor lab; the specific issue part discusses several problems during the implementation of automatic stimulator; the approach section describes in detail the techniques utilized to implement the automated stimulator, and shows all the jobs done step by step in the format of time line; at last a conclusion is given.

## *BACKGROUND*

The crayfish tail receptor stimulator in this project is a replacement of original manual stimulator in the crayfish stretch receptor lab, so it is important to first understand the content of the lab in order to understand the role of such manipulator. In the lab, students need to record extracellularly from the nerve which carries sensory information from the muscle receptor organs (MROs) to the central nervous system while curling the tail to stimulate these receptors. The recording results allow students to determine the stimulus-response properties of the MROs and to measure the adaptation rate of the MROs. Some basic features of sensory systems will be observed in this exercise. According to the lab manual, the lab mainly contains dissection, recording and main experiment.

After the dissection, which makes the crayfish tail ready for the lab, attach the thread to a manipulator so that changing the horizontal position of the manipulator curls the tail. The tail is pinned in the dish at the anterior end. Then choose a nerve in a posterior segment and advance the suction electrode toward the nerve branch (usually no activity in the nerve until the tail is curled). Set the oscilloscope input to AC coupling and 2 to 5 ms/div. When the tail is curled, superficial extensor muscle which contains the MROs should be seen under high magnification. Then set the oscilloscope to a fast time base (0.5 to 2.0 ms/div) and stretch the tail.

When comes to the main experiment, first use the manipulator to slowly curl the abdomen and observe the action potentials in the nerve. Increase the stretch. Apply a moderate amount of curl to the tail, and then tap the thread lightly to give a sudden brief stretch. Two sizes of action potential should be seen in response to this stimulus.

After that stimulus response of MRO is tested. Relax the tail just to the point at which MRO is no longer active. Curl it again by a measured amount (using the scale on the manipulator). Measure the receptor response. Then measure the response with several different amounts of stretch and graph the relationship between receptor response and units of stretch.

At the end of main experiment, MRO's adaption is tested. Adaptation is a basic feature of sensory systems which causes the phenomenon that the receptor is most active just after the full stretch is achieved and that the action potential firing then slows down. To do the test, quantify adaptation by recording 20 to 30 s of the response after a stretch. Then the adaptation rate will be the slope of the line that results from this transformation if graphing the instantaneous firing data as log frequency vs. time.

The instruments of this lab are shown in the following figure. Note that the one in red circle is the manual crayfish tail receptor stimulator.

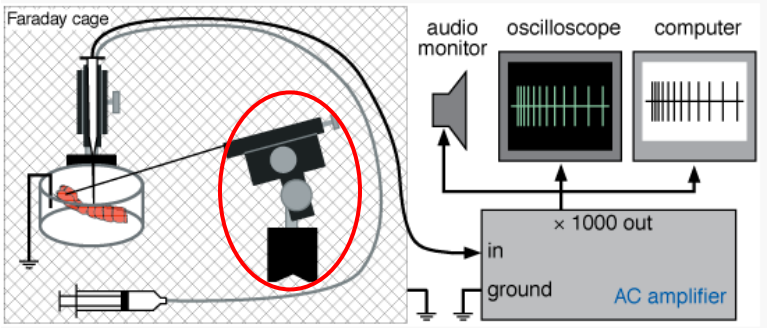



Figure 1 Crayfish lab equipment

## *DESIGN AND IMPLEMENTATION*

### High Level Design

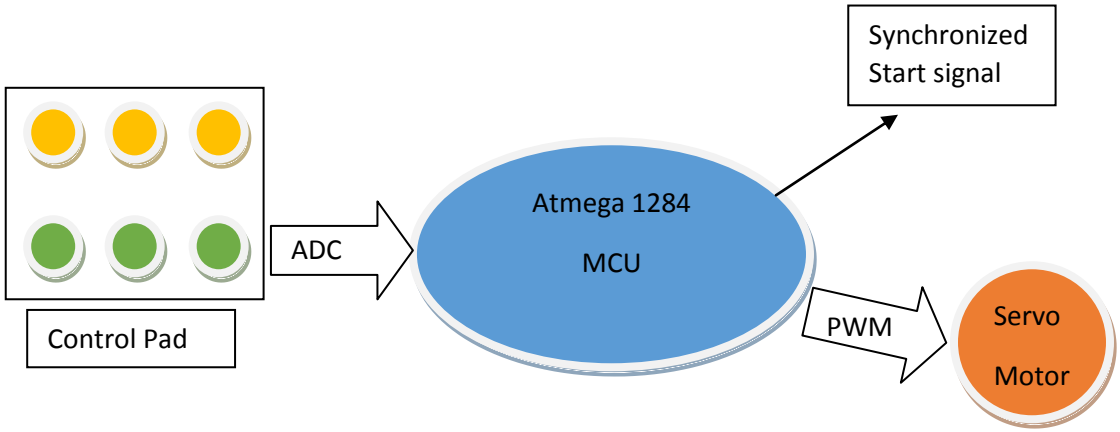



Figure 2 Device high level design

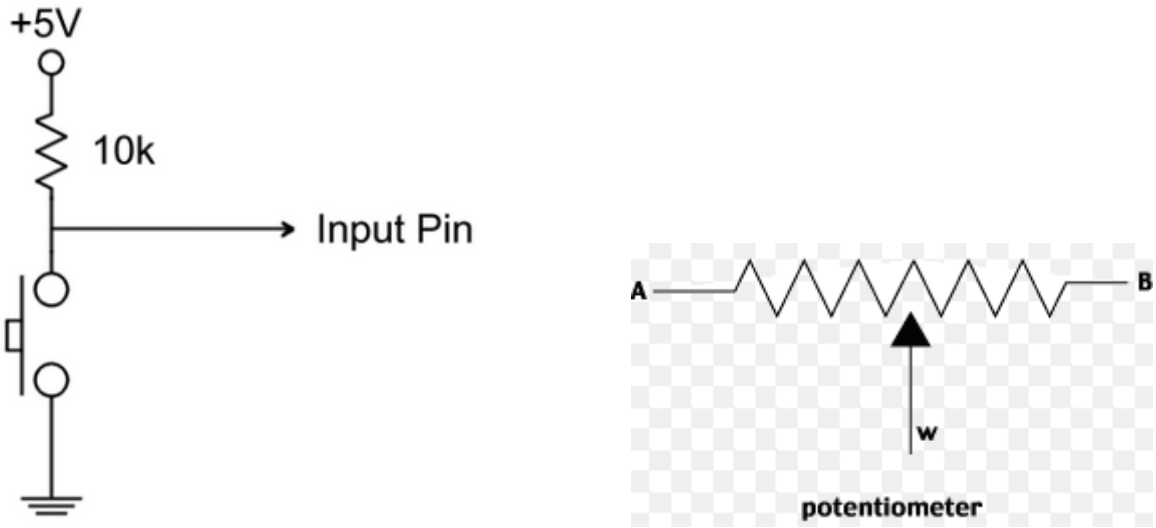
From the high level point of view, our project has three main parts: control pad, microcontroller, and servo motor. The Control pad includes one power switch, three push buttons and three knobs. Three push buttons control the "initial" position set, "final" position set and "pull" action. Correspondingly, three knobs control the "initial", "final" position, and pull speed. For the position knobs, they will affect the servo motor only when the related position set is pressed.

The brain of the design is the ATmega microcontroller. It takes the input from the control pad by converting the analog signal to digital through the on chip ADC, and transmits the computed output to the servo motor. All the logic and computation were done inside the microcontroller, and it generated a synchronized start signal before it sent out the PWM outputs.

The last part is the servo motor, which determined how far we pull the crayfish tail from the initial position. The servo motor took PWM signal and rotated certain angle based on the inputs. This servo motor was attached with a thread, which stretched the crayfish tail.

### Hardware Design

From the control pad, we have two types of hardware, push button and potentiometer. The push button and potentiometer circuitry is shown in Figure 3. The critical issue for a push button is that we have to do a debouncing detection. This part is done in a finite state machine in software. As the figure (a) shows, we added a pull up resistor to the input pin and connected the push button to ground. In this way, we could detect a low signal when we pushed the button. For the potentiometer, we connected one side to Vdd and the other side to ground. When we turn the knob, the input value will be an analog voltage between zero and Vdd. The on chip ADC will convert the analog signal to digital data for us.




(a) (b)

Figure 3 (a) Push button circuit (b) potentiometer circuit

For the servo motor connection, we had three wires to be connected, Vdd, ground, and PWM signal. As is shown in the Figure 4, we connected the orange to PWM signal, red to Vdd, and brown wire to ground.

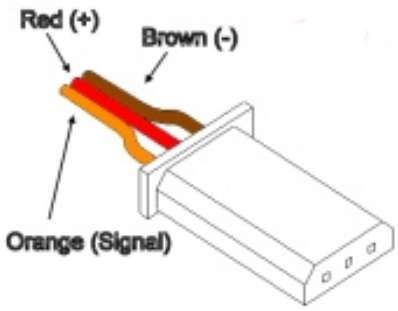



Figure 4 Servo motor connection

**Software Design**

We will program the two buttons with two inputs to start and reset the program. These knobs will control two potentiometers and the value will be read though an ADC on the MCU. These digital signals will set the PWM parameters, which control the servo position and speed. We also need to generate a synchronized start signal when the start button is pushed, so that we can tell the oscilloscope to start capture at that time.

The finite state machine of the design is shown in the above figure (Figure 2). The detailed operations are:

Initial state: This is the first state when we start the machine. In this state we connect the first ADC which is used to generate the 8 bits number from knob to OCR0A, which is used to set the initial position. Additionally, in this state we need to check whether the other two buttons are pushed. If one of them is pushed, then it needs to switch to another state, otherwise just keeps in this state.

State 1: This state is used to debounce the first button. If the first button is pushed, then go step into initial state, otherwise go step back to the previous state.

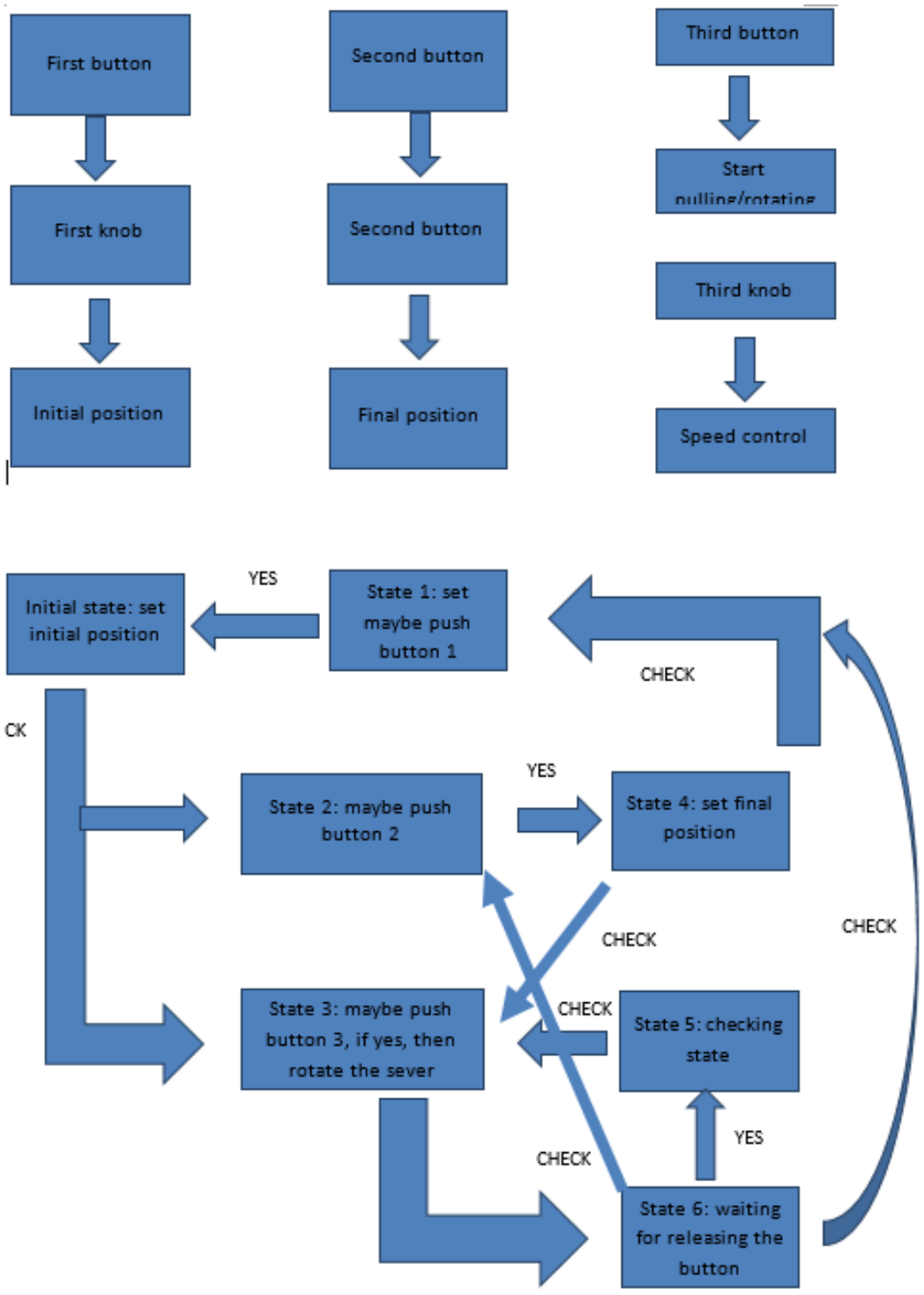



Figure 5 Software finite state machine

State 2: This state is used to debounce the second button. If the second button is pushed, then go step into state 4, otherwise go step back to the previous state.

State 3: This state is used to debounce the third button. If the third button is pushed, then go step into state 6, otherwise go step back to the previous state. One more thing which need to be done in this state is that we need to figure out the difference between the initial and final position "Delta". Then we can set step size which is equal to the rotate speed of the server. What's more, according to Delta and step size we can finish rotating the server.

State 4: In this state we have confirmed that the first button is pushed. Then we can rotate the knob to control the angle of the server as the initial position.

State 5: In this state what we plan to do is to check which button is pushed, then it will go step into the corresponding state to debounce every button. If none of the three button is pushed, then the state machine will stay in this state.

State 6: This state is used to check whether the third button is released after pushing, otherwise it will go several steps even you only push once. So if as long as the button is held, it will always stay in this state. And only if you release the button, it will go step into state 5 to change the state to whatever you want.

**Mechanical Design**

The mechanical design of this project mainly focuses on the box and the control panel of the device. We used a 12cm x10cm x 6cm box in our design, because it is large enough to hold our circuit boards in it. In addition, considering the convenience of manipulating the device during the labs, we chose the enclosure of this size with a slanted top. The whole control panel is designed on that top. The control panel is shown in Figure 6.

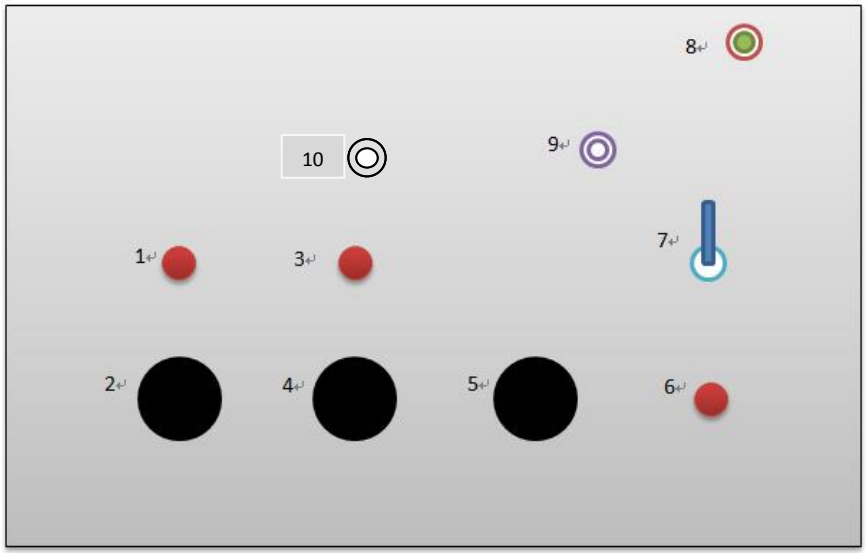



Figure 6 Control panel design

Button 1 is the initial position button. It sets the initial position and signals the servo to rotate back to the initial position. Knob 2 is used to adjust the initial position. Every time it is used, button 1 should be pressed afterwards to set the initial position.

Button 3 and knob 4 are used together to set the stop position. Knob 4 is used to adjust the stop position, and button 3 is used to set the final position.

Knob 5 is a speed knob that changes the servo rotation speed. It is able to be rotated at any time during manipulation. Button 6 is the start button that notifies the servo to rotate as set. Every time it is pressed, the servo will rotate by one step. The step size is set by the control units 1, 2, 3 and 4 in above figure.

Control unit 7 is the power on/off button of the device. When it is on, the light (control unit 9) will be on to notify the user. Control unit 9 is a socket that connects the servo motor. Since it is convenient to put the motor onto an iron holder, we designed this socket here to make the motor easily connected to the control device. Interface 10 is designed to connect to the oscilloscope. When the button 6 (pull start) is pressed, an signal is sent via this interface to notify oscilloscope to record result signals.

The actual device control panel is shown in Figure 7, and the internal board connection is shown in Figure 8.

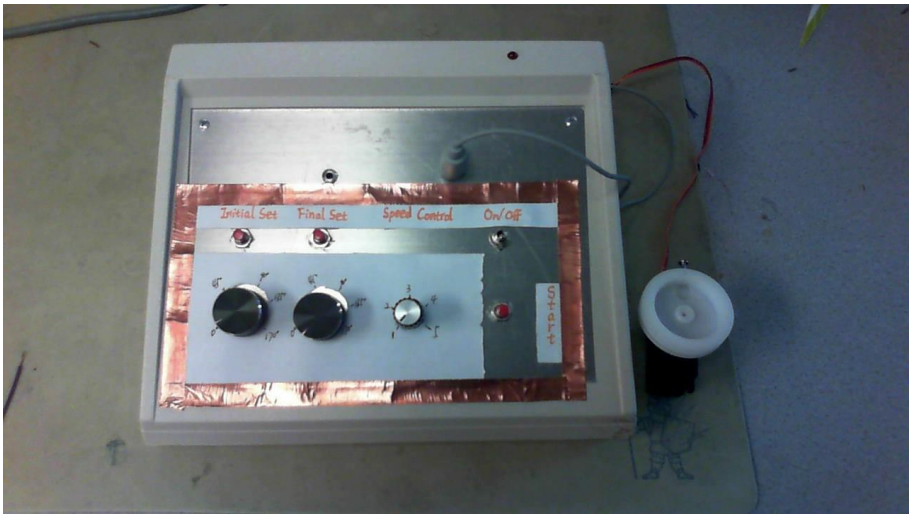



Figure 7 Actual device control panel

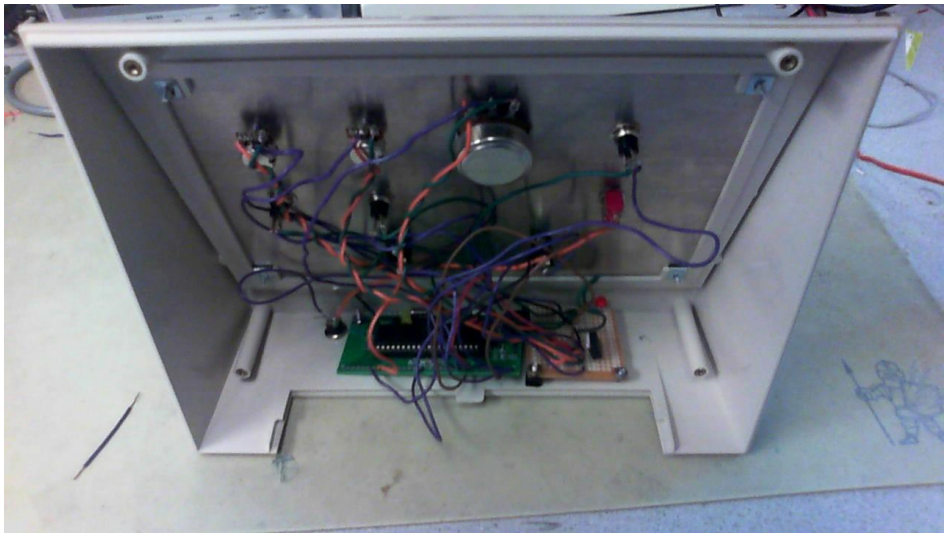


Figure 8 Internal circuit board connection

## *RESULT AND ANALYSIS*

The whole final device is shown in Figure 9.

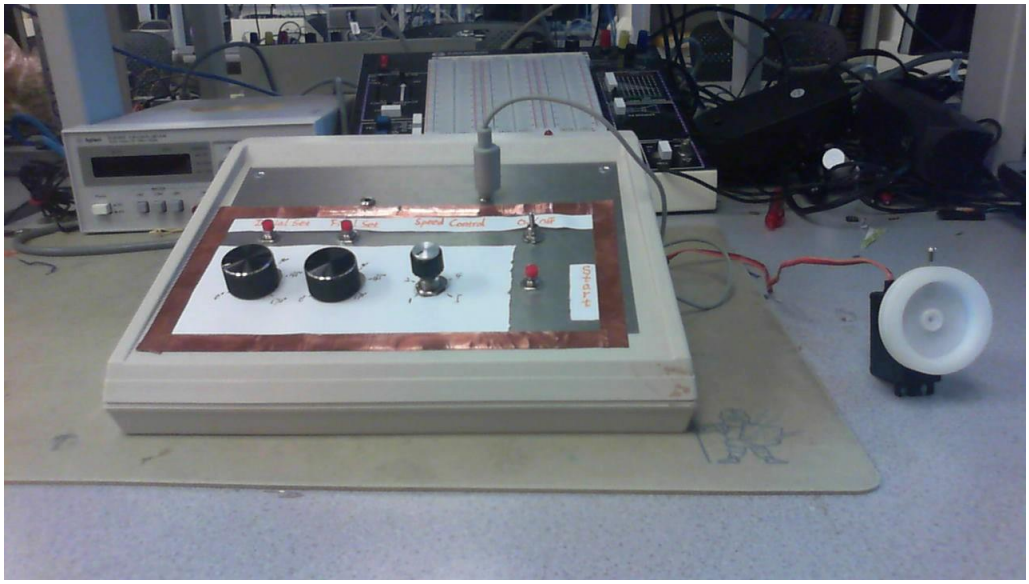



Figure 9 Overall device design

The device above is a stimulator that stretch the crayfish tail. Together with the nerve sensor, we are able to get the crayfish tail nerve reaction shown in Figure 10.

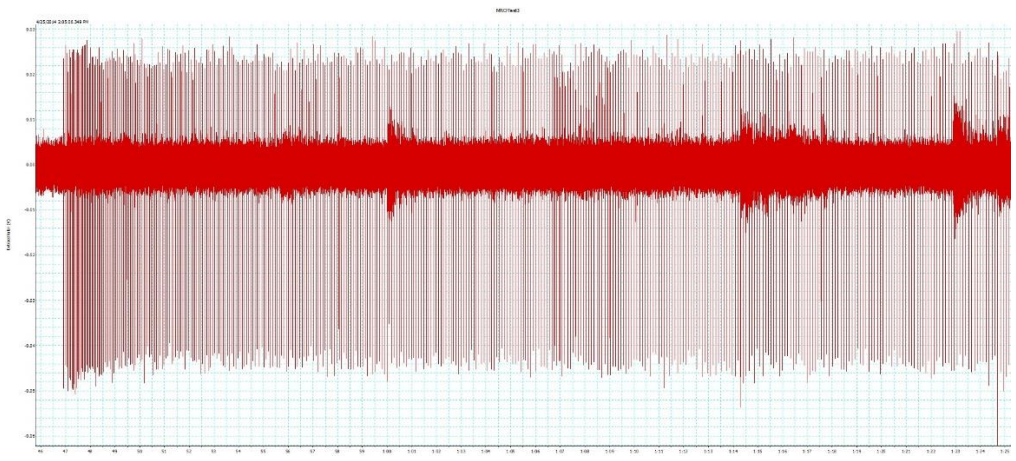


Figure 10 Crayfish tail nerve reaction

The reaction shown in Figure 10 is recorded in voltage in the time domain. The red dense part in the middle of the figure records the noise of the lab environment. The signals with larger amplitude are the nerve reaction signals when the crayfish tail is dragged. The above figure is the reaction result after one drag. At the beginning the reaction signal is dense, and after a short time

the signal frequency reduces slightly to a constant value. Such variation is due to the fact that the nerve is able to adapt to the stretch.

Figure 10 indicates that our designed device performed well in crayfish tail receptor lab. Such a lab requires little noise interference, and our device works well in noise avoidance, which can be seen from the fact that the reaction signal's amplitude is much larger than noise's amplitude in Figure 10. In addition, the reaction is very clear and strong, so it is very easy for researchers to observe and to do further analysis.

## *Task division*

Zequn Huang: Software control algorithm, motor speed control hardware design.

Ningning Ding: Software state machine design, software testing

Jiachen Hu: Mechanical frame design, stimulator position control.

Work together: Overall testing and improvement.

## *Tentative Parts Cost List*

| Part | Model | Cost | Source |
|---|---|---|---|
| Servo Motor | Generic High Torque Full Rotation (ROB-09347) | $11.95 | https://www.sparkfun.com/products/9347\ https://www.sparkfun.com/products/10189 |
| Microcontroller | Custom PC board ATmega 1284 | $9.00 | Get from Cornell ECE 4760 Lab |
| Mechanical Controls | Servo Motor mount Mechanical frame | $3.00 | Scrounge from building materials |

**Timeline:**

| Time Schedule | Task |
|---|---|
| Week1 (10/14-10/20) | Start-up Group meeting |
| Week2 (10/21-10/27) | Come up with design ideas |
| Week3 (10/28-11/3) | Background research, design specification |
| Week4 (11/4-11/10) | First draft design proposal, order parts |
| Week5 (11/11-11/17) | Build mechanical framework, test servo |
| Week6 (11/18-11/24) | Generate servo speed control algorithm<br>Project presentation |
| Week7 (11/25-12/1) | Design “start” and “reset” servo control<br>Test existing functions |
| Week8 (12/2-12/8) | Final proposal and presentation |
| 12/9/2013-1/26/2014 | Winter Break |
| Week9 (1/27-2/2) | Generate stimulator position control algorithm and hardware design |
| Week10-11 (2/3-2/16) | Testing and problem solving |
| Week12-13 (2/17-3/2) | Design Finalization and Prototype |
| Week14 -16(3/3-3/23) | Final report |

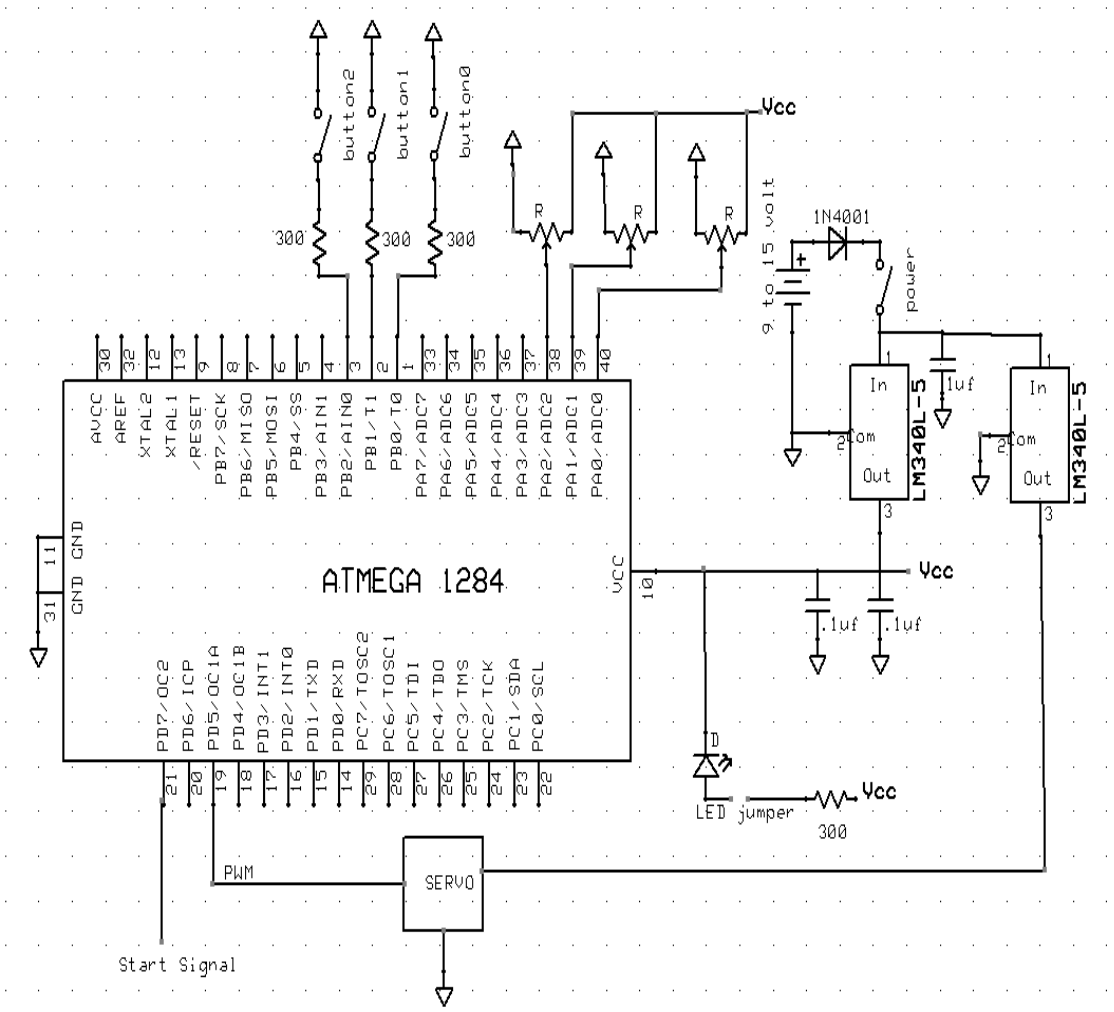
## *CONCLUSION*

The crayfish tail receptor stimulator designed in this project is an automatic manipulator that used to curl the crayfish tails in crayfish stretch receptor labs for students and in other study experiments for researchers. It is aimed to pull the crayfish tail more stably and more reliably, so that nerve responses can be better observed and studied. Several issues including old receptor stimulator, new stimulator system and laboratory equipment were discussed in the proposal, with advantages and shortcomings in first two issues analyzed. Then the approach to the project aim was shown, and each member’s tasks were specified. At last tentative components with detailed information were listed, and the planned timeline of the project was worked out. When finishing

this project, an automated manipulator is expected to provide students and researchers with more convenient experiment experiences.

# Appendix I

## Schematic



# Appendix II

## Code

```
#include <stdio.h>
#include <util/delay.h>
#include <stdlib.h>
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>

/* CPU frequency */
#define F_CPU 16000000UL

// ISR
#define SUSPEND cli();
#define RESUME sei();

// serial communication library
#include "uart.h"
#include "uart.c"

// UART file descriptor
// putchar and getchar are in uart.c
FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar,
_FDEV_SETUP_RW);
```

```
// the usual
#define begin {
#define end }

#define READ(U, N) ((U) >> (N) & 1u)
#define SET(U, N) ((void)((U) |= 1u << (N)))
#define CLR(U, N) ((void)((U) &= ~(1u << (N))))
#define FLIP(U, N) ((void)((U) ^= 1u << (N)))


#define Pushedset 1
#define MaybePushset 2
#define Pushedfinal 3
#define MaybePushfinal 4
#define Pushedpull 5
#define MaybePushpull 6
#define Waiting 7
#define mayberel 8

void initialize(void);

//interupt for ADC convertion
ISR(ADC_vect)
begin
        ADCSRA |=1<<ADSC;
```

```
end


int main(void)
begin
initialize();
int step;
int initial = 2226, final = 2226, delta = 0;
int PushState = Pushedset;
int PreState = Pushedset;
int n=0;

while(1)
begin

switch (PushState) //state machine begin
        begin
                case MaybePushset:    //debounce for initial setting state
                        if (~PINB & 0x01)
                                PushState = Pushedset;
                        else
                                PushState = PreState;
                        break;
                case Pushedset:      //give the ADC value to PWM output
                        begin
                                //ADMUX = "ch0";
                                ADMUX = 0x60;
```

```
  //map the ADC value to 770~2260
        OCR1A = (int)(ADCH * 5.71) +770;
            initial = OCR1A;
            delta = final - initial;
            if (~PINB & 0x04)
                {
                    PushState = MaybePushpull;
                    PreState = Pushedset;
                }
            else if (~PINB & 0x02)
                {
                    PushState = MaybePushfinal;
                    PreState = Pushedset;
                }
            else
                PushState = Pushedset;
        end
        break;
case MaybePushfinal:  //debounce for final position setting
        if (~PINB & 0x02)
            PushState = Pushedfinal;
        else
            PushState=PreState;
        break;
case Pushedfinal:    //give another ADC value to PWM output
        begin
            //ADMUX = "ch1";
```

```
                ADMUX = 0x61;
                OCR1A = (int)(ADCH * 5.71) +770;
                //fprintf(stdout,"%d  \n\r",ADCH);
                final = OCR1A;
                delta = final - initial; //calculate the distance
                if (~PINB & 0x01)
                        {
                                PushState = MaybePushset;
                                PreState = Pushedfinal;
                        }
                else if (~PINB & 0x04)
                        {
                                PushState = MaybePushpull;
                                PreState = Pushedfinal;
                        }
                else
                        PushState = Pushedfinal;
        end
        break;
case MaybePushpull:  // debounce for pulling state
        _delay_ms(10);
        if (~PINB & 0x04)//If pulling, then go step to pull according to the speed
        {

                PushState = Waiting;
                ADMUX = 0x62;
```

```
            SET(PORTD,7);
            _delay_ms(12);
            CLR(PORTD,7);

            step = (int)((255-ADCH)/50+1);
            n = (int)delta/(-step);
            if(n<0)
            {
                  n=-n;
                  step = -step;
            }
            while(n!=0)
            {
                  OCR1A = OCR1A - step;
                  n--;
                  if(OCR1A >= 2226)  //protect the servo from overflow
                        OCR1A = 2226;
                  else if(OCR1A <= 770)
                        OCR1A = 770;
                  _delay_ms(3);

            }


}
else
            PushState=PreState;
```

```
        break;
case Pushedpull:    //the state to transfer to another state
        begin

                delta = final - initial;
                if (~PINB & 0x01)
                        {
                                PushState = MaybePushset;
                                PreState = Pushedpull;
                        }
                else if (~PINB & 0x02)
                        {
                                PushState = MaybePushfinal;
                                PreState = Pushedpull;
                        }
                else if (~PINB & 0x04)
                {
                        PushState = MaybePushpull;
                        PreState = Pushedpull;
                        }
                else
                        PushState = Pushedpull;


        end
        break;

case Waiting:    //wait until release the button
```

```
                    if (~PINB & 0x04)
                            PushState = Waiting;
                    else
                            PushState = Pushedpull;
                    break;

        end

end
end

void initialize(void)
begin
        // PortA ADC input, PortD 4,5 PWM output
        DDRA = 0;
        DDRD =(1<<PIND7) | (1<<PIND4) | (1<<PIND5);
        DDRB = 0;
        PORTB = 0xff;
        //USE TIMER 1 for MOTOR CONTROL
        // 8 prescaler to slow down the PWM
      // divide by 8  and set to Phase and Frequency Correct
        TCCR1B = 2 | (1<<WGM13);   // turn on PWM
        // turn on fast PWM and OC1A outputs
        TCCR1A = (1<<COM1A1) | (1<<COM1B1) ;
        OCR1A = 1500 ; // controls actual movement of servo
        ICR1 = 20000; // frequency = fclk/(2*prescaler*ICR1) = 50Hz
        //can change OCR1A between 770 and 2230 to obtain 1-2ms high pulses
```

```
        //Configure the ADC using ADC0
        ADCSRA |= (1<<ADPS2) | (1<<ADIE) | (1<<ADEN); // ADC prescaler is 16
        ADMUX = (1<<ADLAR) | (1<<REFS0);

        // turn on all ISRs
        sei() ;

        ADCSRA |= 1<<ADSC;

        //init the UART -- trt_uart_init() is in trtUart.c
        //trt_uart_init();
   uart_init();
        stdout = stdin = stderr = &uart_str;
        fprintf(stdout,"\n\r TRT 9feb2009\n\r\n\r");

end
```