Drosophila Automated Olfactory Training and Testing System for Associative Learning

By

Hui Jiang

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

May, 2015

Nashville, Tennessee

Approved:

Bharat L. Bhuva, Ph.D.

Kendal S. Broadie, Ph.D.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Bharat Bhuva. I could not have accomplished this project without your guidance and support. I am also very grateful to the Broadie Lab, Dr. Kendal Broadie, Dr. Cheryl Gatto and Eriny Hanna. Their valuable comments led to important improvements in this thesis. Moreover, I would like to thank all my friends, and specially Zach, for their support.

Last but not least, I would like to express my immense gratitude to my father, my mother and my brother for their love and support during my graduate studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter I

Introduction

Purpose and Goals

The concept of automated systems has always fascinated me. It began from an early age when I saw products being automatically wrapped by a machine on my first factory trip. As my fascination with automated systems continued, I studied electronic circuits and programming.

I joined the Broadie lab graduate project team in 2014 with the goal of designing a *Drosophila* automated training and testing system for olfactory associative learning induced with paired electrical shock conditioning. The *Drosophila* genetic system has been used in Pavlovian classical conditioning experiments for over 40 years, greatly improving our understanding of learning formation and memory consolidation [1]. The most common type of *Drosophila* classical conditioning is induced in a population of flies by releasing both a neutral odor cue (conditioned stimulus, CS+) along with an electric shock-aired odor cue (unconditioned stimulus, US) [1]. A second conditioned stimulus (CS-) is then released without the US. During the testing phase, *Drosophila* is simultaneously presented with both CS+ and CS- odors, provided time to choose between the odors with a directional movement and then the population distribution for each odor choice is recorded. This process allows associative aversive, as opposed to appetitive/reward conditioning, to be quantitatively measured without a bias introduced by the innate preference for either of the conditioned stimuli [1].

Conditioned odor learning using electrical shock punishment is a common procedure for testing learning and memory behavioral outcomes in *Drosophila* [2]. The experiment is performed in two phases: the flies are trained in the first phase, and the trained flies are tested in the second phase. During a single training cycle, 100 flies are sequentially exposed to two matched, equally aversive odors (e.g. octanol and methyl-cyclohexanol). The first odor is conditioned by delivery of a series of 10 electric shocks (2.5 seconds each at 80V DC) through a copper grid every 5 seconds (2.5 seconds rest between each shock), with shock-free exposure to the second non-conditioned odor following thereafter. Learning assessments can be done with a single mass training session, whereas long-term memory (e.g. 24 hours later) requires spaced training (7-10 sessions with 15 minutes rest between each) [3]. Afterward, flies are tested for the ability to learn (15 minutes following training) and then remember (24 hours following training) in a T-maze presenting both odors. Learning occurs with pre-existing cellular components not requiring new protein synthesis, whereas 24 hour long-term memory (LTM) requires new protein translation [3]. Manually conducting these learning and memory studies is extremely labor-intensive and subject to operator variation. Developing a *Drosophila* automated training and testing system would greatly increase the scope of studies that could be pursued, increase precision control and allow more consistent reproduction of outcome results. The overall goals of this project are the following:

- Achieve automatic delivery of volatile odorants
- Computer-controlled delivery of electrical shocks
- Include fail-safe tests to ensure that airflow rates and electrical shocks are delivered correctly
- Achieve automated transition experiment phase from training phase to testing phase
- Attain a workable design that can be improved over time

Background

Over many years, several different semi-automated conditioning systems have been using for olfactory training and testing of *Drosophila*, contributing a great deal to their study, especially the identification of genes involved in learning formation and memory consolidation [4]. However, costly custom-built parts were used to assemble these machines, for which detailed designs are not readily available. One semi-automated system shows the detailed design method [4]. They use an interface (USB-6501, National Instruments Co.) that connects the stimulator and solenoid valves to a PC. The 5V-trigger signals transmitted from USB-6501 are amplified to 12V by a relay (PS7113-1A-A, NEC Electronics) to control the solenoid valves. The LabVIEW software package is utilized to control the gating of solenoid valves that allows air to go through only one of three liquids and into the training tube [4]. A semiconductor-based odor sensor is employed to monitor odor concentration in the training tube (TSG2620, Figaro Co.). The outputs of the odor sensor were recorded with a data acquisition device (USB-6212, National Instruments Co.).

Contributions

The main contributions of this thesis are the followings:

- Develop a *Drosophila* automated training and testing system that is fully automated, including transferring from the training phase to the testing phase.
- Include fail-safe tests to ensure that odorants, air flow rates and electrical shocks are delivered correctly and consistently.
- Use general-purpose parts that are commercially available and keep the each *Drosophila* automated training and testing system cost below $700.

Structure of the Thesis

The rest of this thesis is structured as follows. In Chapter 2, the design theory of the *Drosophila* automated training and testing system is detailed. First, the requirements, specifications and design considerations are stated and explained. Then, the two theory methods are summarized and illustrated. Subsequently, in Chapter 3, the electrical and mechanical design components are specified. Next, in Chapter 4, the software development is discussed. Experimental results are presented in Chapter 5. Conclusions and future development are summarized in Chapter 6.

Chapter II

Two Design Theories of *Drosophila* Automated Training and Testing System

The first part of this chapter describes the requirements, specifications and design considerations of the *Drosophila* automated olfactory training and testing system developed in the work presented in this thesis. This system was developed at Broadie Laboratory of Vanderbilt University. In the second part, the two theory methods are summarized and illustrated.

General requirements, specifications and design considerations

The *Drosophila* automated training and testing system was designed to automate the model presented in this thesis that is declared in chapter I. Therefore, one of the main requirements was to design a fully automated system, reducing or eliminating the need for human interaction. Servomotor and linear actuator controls have been the two main methods in which research has been conducted to achieve a self-moving experiment phase, developed at the Broadie Laboratory of Vanderbilt University.

The automated system was designed for olfactory learning and memory, therefore the components must be made of anti-corrosive materials capable of resisting odorants (e.g. 3-octanol and 4-methylcyclohexanol). Moreover, studies are done at high levels of relative humidity (>90%), therefore the components must be made of non-rusting materials. The automated system should also ensure that time periods for all functions of the system are correct, like the model.

Concepts of Drosophila Rotating/Linear Automated Training and Testing System

Two methods have been attempted to achieve a fully automated system. The first I named the "Rotating Automated Training and Testing System" (RATTS), which uses two servomotors to control two of the 3D parts to achieve the model requirements. I named the second automated system "Linear Automated Training and Testing System" (LATTS), which uses a linear actuator to move only one of the 3D parts to achieve the paradigm requirements. RATTS and LATTS used two different concepts to achieve automated experiment phase change. Therefore, the 3D parts design of RATTS and LATTS are completely different. A detailed description of each method is given in the following subsections.

Concepts of Drosophila Rotating Automated Training and Testing System

RATTS's 3D designed parts need to automatically transform the experiment phase from the training phase to the testing phase. This means the RATTS needs be able to move the flies from the training tube to the T-maze's center, which is presented with both CS+ and CS- [1]. Figure 1 illustrates the rotating automated training and testing system's approach for learning and memory behavior.

There are two servomotors in the RATTS, one of the servomotors controls the cylinder and the other controls the screen filter. For the learning training phase, after *Drosophila* trained in the training tube, the screen filter rotates clockwise 45 degrees. Then, the air pump pushes the *Drosophila* from the training tube to the center of the cylinder. Screen filter rotates counter-clockwise 45 degree to lock the *Drosophila* in the center of the cylinder. During the testing phase, the screen filter rotates clockwise 45 degrees and the cylinder rotates clockwise 90 degree. This exposes *Drosophila* to both CS+ and CS- odors. After 2 minutes, for choosing time, the cylinder rotates counter-clockwise 45 degrees to lock the *Drosophila*, which have already chosen one of the odors. This also locks the *Drosophila* which haven't made an odor choice in the center of the cylinder. Finally, the cylinder rotates clockwise 45 degrees, in order to determine the number of the *Drosophila* that stayed in the center of the cylinder.

For memory training and testing, after training the *Drosophila* in the training tube, instead of the screen filter rotating clockwise 45 degrees, the screen filter rotates counter-clockwise 45 degrees and air pushes the *Drosophila* from the training tube to the resting tube. Then the screen filter rotates counter-clockwise 45 degrees to lock the flies in the resting tube.

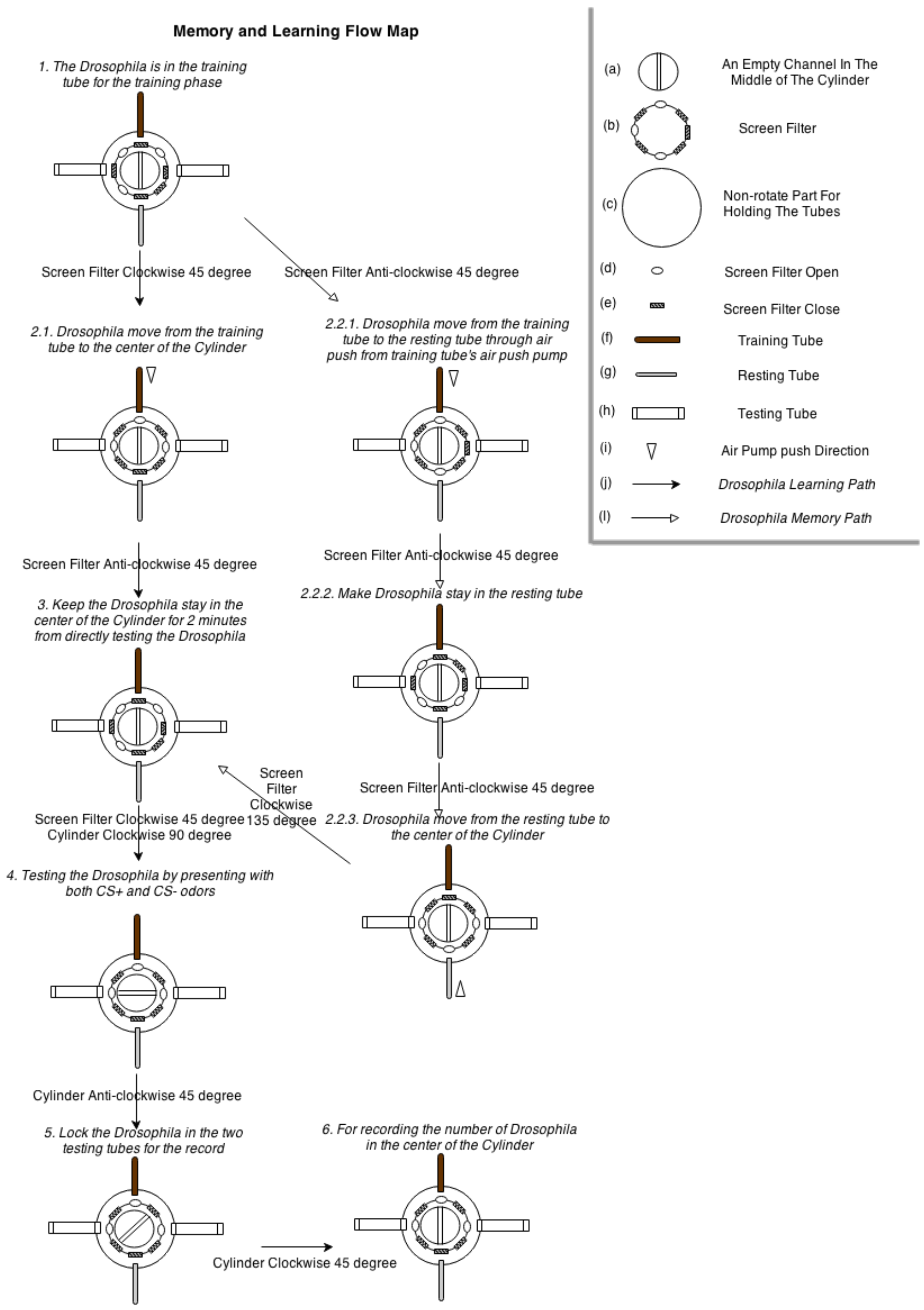

Memory and Learning Flow Map
1. The Drosophila is in the training tube for the training phase
Screen Filter Clockwise 45 degree
Screen Filter Anti-clockwise 45 degree
2.1. Drosophila move from the training tube to the center of the Cylinder
2.2.1. Drosophila move from the training tube to the resting tube through air push from training tube's air push pump
Screen Filter Anti-clockwise 45 degree
Screen Filter Anti-clockwise 45 degree
3. Keep the Drosophila stay in the center of the Cylinder for 2 minutes from directly testing the Drosophila
2.2.2. Make Drosophila stay in the resting tube
Screen Filter Anti-clockwise 45 degree
Screen Filter Clockwise 135 degree
Screen Filter Clockwise 45 degree
Cylinder Clockwise 90 degree
2.2.3. Drosophila move from the resting tube to the center of the Cylinder
4. Testing the Drosophila by presenting with both CS+ and CS- odors
Cylinder Anti-clockwise 45 degree
5. Lock the Drosophila in the two testing tubes for the record
6. For recording the number of Drosophila in the center of the Cylinder
Cylinder Clockwise 45 degree
(a) An Empty Channel In The Middle of The Cylinder
(b) Screen Filter
(c) Non-rotate Part For Holding The Tubes
(d) Screen Filter Open
(e) Screen Filter Close
(f) Training Tube
(g) Resting Tube
(h) Testing Tube
(i) Air Pump push Direction
(j) Drosophila Learning Path
(l) Drosophila Memory Path

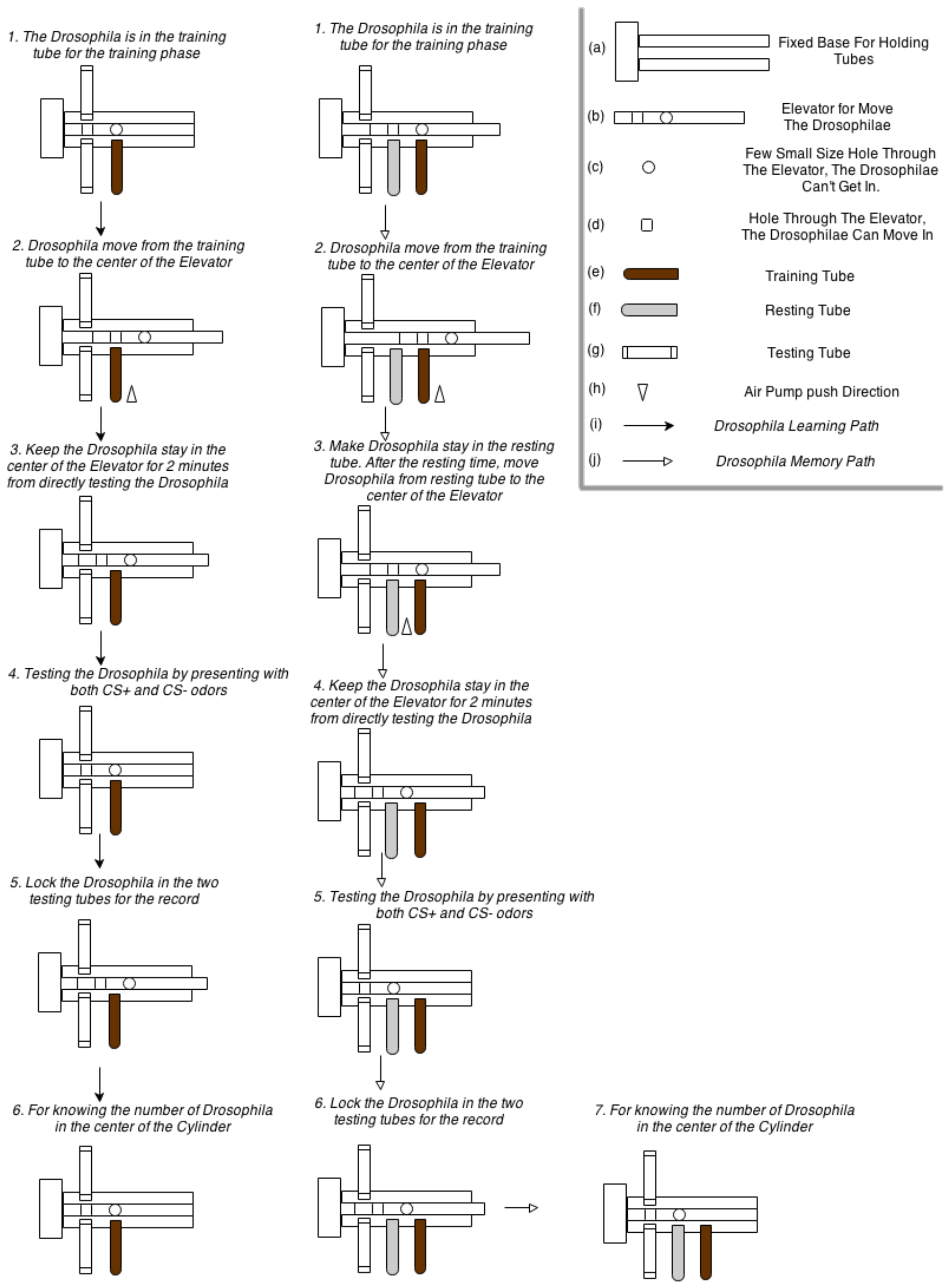
Figure 1. Rotating automated training and testing system flow map. The 3D printed parts mainly include the cylinder, the screen filter, and non-rotating part. One servomotor controls the cylinder and the other servomotor controls the screen filter.

After 24 hours, the screen filter rotates counter-clockwise 45 degrees again. Then, the air pump pushes the *Drosophila* from the resting tube to the center of the cylinder. After that, the screen filter rotates counter-clockwise 45 degrees to lock the flies in the center of the cylinder. The next process will be the same as the learning testing phase procedure. The flow map as shown in the Figure 1 explains all of these processes.

Concepts of Drosophila Linear Automated Training and Testing System

LATTS uses a linear actuator instead of two servomotors to automatically transfer the *Drosophila* from the training phase to the testing phase. Therefore, LATTS's 3D design concepts are different from RATTS's. Figure 2 illustrates the linear automated training and testing system's approach for learning and memory behavior in *Drosophila*.

In LATTS, we use a linear actuator to control an elevator. For the *Drosophila* learning experiment, first we train the flies in the training tube. Second, we use the linear actuator to move the elevator to make sure the hole opening position directly faces the training tube. Using an air pump, we push the *Drosophila* from the training tube into the elevator. Third, the *Drosophila* stay in the elevator for two minutes by adjusting the elevator's position to closed on both sides. Fourth, we move the elevator's position to let the open hole position directly face the testing tube. Fifth, *Drosophila* in the two testing tubes are collected and counted to obtain learning scores. Finally, after obtaining learning scores, we count the number of *Drosophila* that stay in the elevator.



1. The Drosophila is in the training tube for the training phase
2. Drosophila move from the training tube to the center of the Elevator
3. Keep the Drosophila stay in the center of the Elevator for 2 minutes from directly testing the Drosophila
4. Testing the Drosophila by presenting with both CS+ and CS- odors
5. Lock the Drosophila in the two testing tubes for the record
6. For knowing the number of Drosophila in the center of the Cylinder
1. The Drosophila is in the training tube for the training phase
2. Drosophila move from the training tube to the center of the Elevator
3. Make Drosophila stay in the resting tube. After the resting time, move Drosophila from resting tube to the center of the Elevator
4. Keep the Drosophila stay in the center of the Elevator for 2 minutes from directly testing the Drosophila
5. Testing the Drosophila by presenting with both CS+ and CS- odors
6. Lock the Drosophila in the two testing tubes for the record
7. For knowing the number of Drosophila in the center of the Cylinder
(a) Fixed Base For Holding Tubes
(b) Elevator for Move The Drosophilae
(c) Few Small Size Hole Through The Elevator, The Drosophilae Can't Get In.
(d) Hole Through The Elevator, The Drosophilae Can Move In
(e) Training Tube
(f) Resting Tube
(g) Testing Tube
(h) Air Pump push Direction
(i) Drosophila Learning Path
(j) Drosophila Memory Path

Figure 2. Linear automated training and testing system flow map. The 3D printed parts mainly include the base and the elevator. A linear actuator controls the elevator, in order to move the elevator to the specific location.

Chapter III

Electrical and Mechanical Design Components

A schematic diagram of RATTS and LATTS are shown in Figure 3. Both machines consist of 8 solenoid valves, 3 one-way check valves, 5 odor cups, 1 air pump, 1 vacuum pump, 3 thermal mass flow sensors, 1 odor sensor, different types of fittings and different sizes of tubing. Eight solenoid valves act like switches to decide when the odor goes through the tubing and when the air pump pushes the *Drosophila*. Three one-way check valves are used to prevent the different odors from contaminating the odor cup's odor. One air pump provides force to move *Drosophila* from the training tube or resting tube to the center of cylinder/elevator. Three thermal mass flow sensors exist to ensure the airflow rates are kept on target. One odor sensor ensures correct odors delivery. Our solenoid valve is 1/4 npt (national pipe thread taper) female two-port solenoid valve (see chapter III). If we utilize 1/4'' OD (outer diameter) x 0.170'' ID (inner diameter) tubing connected with the solenoid valve. The solenoid valve needs the 1/4'' OD x 1/4 npt male fitting. The 1/4'' OD x 1/4 npt male fitting didn't show in the Figure 3.

Electronics and sensors

The electronics implemented in the RATTS/LATTS can be divided into two main blocks; electronics for controlling and electronics for data logging. Figure 4 shows a schematic picture of electronics involved in the design of the RATTS and LATTS. A detailed description of each electronics block is given in the following titled subsections.

<u>The Microcontroller – Arduino Uno ATMega</u> 328 [5]

The microcontroller was the Arduino Mega 328, selected after careful review of several models. Considering that this single part is the heart of the project, we selected it with special consideration of characteristics like connectivity, available I/O, and board power integration. The microcontroller offered all the features we needed to develop the RATTS and LATTS, with the characteristics we desired. It is based on Atmel's ATMega 328 microcontroller. The board comes with 14 digital input/output pins, 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, and a reset button. The operating voltage of the microcontroller is 5V, with an input voltage of 7 to 12V.

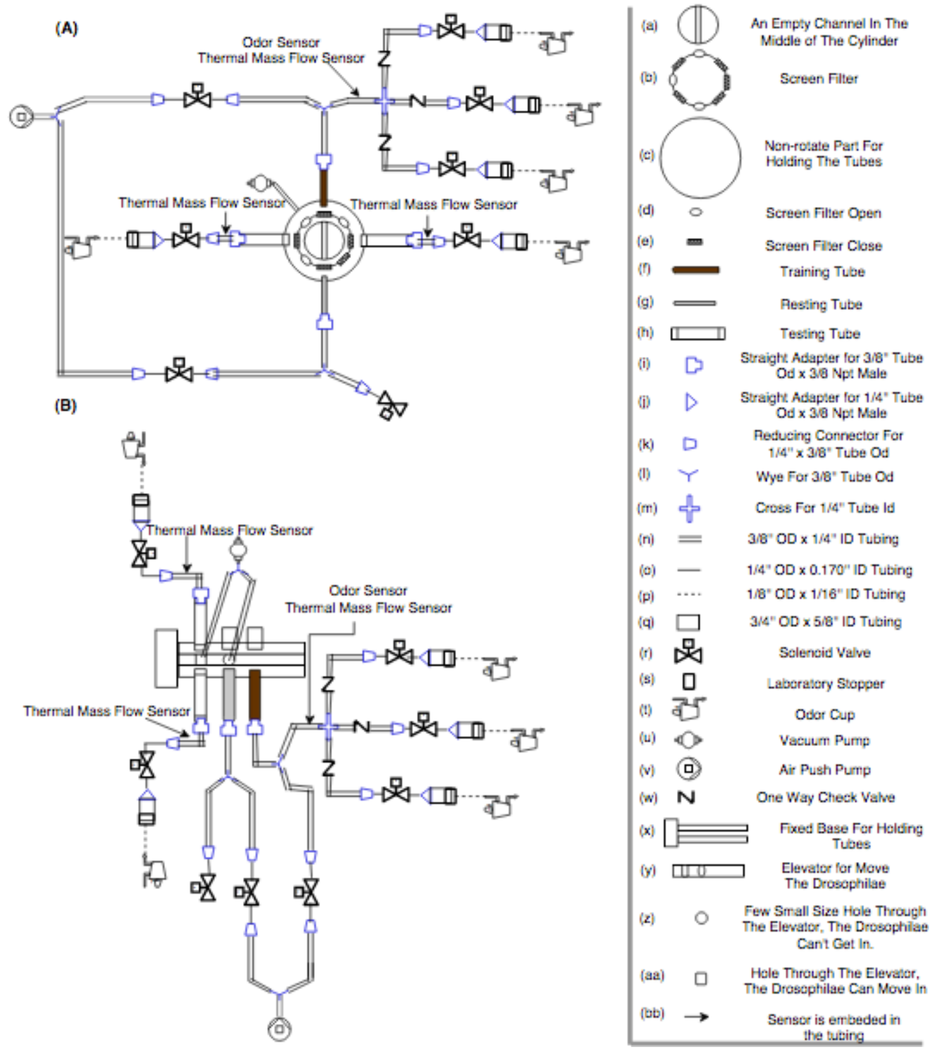

(A)
Odor Sensor
Thermal Mass Flow Sensor
Thermal Mass Flow Sensor
Thermal Mass Flow Sensor
(B)
Thermal Mass Flow Sensor
Odor Sensor
Thermal Mass Flow Sensor
Thermal Mass Flow Sensor
(a) An Empty Channel In The Middle of The Cylinder
(b) Screen Filter
(c) Non-rotate Part For Holding The Tubes
(d) Screen Filter Open
(e) Screen Filter Close
(f) Training Tube
(g) Resting Tube
(h) Testing Tube
(i) Straight Adapter for 3/8" Tube Od x 3/8 Npt Male
(j) Straight Adapter for 1/4" Tube Od x 3/8 Npt Male
(k) Reducing Connector For 1/4" x 3/8" Tube Od
(l) Wye For 3/8" Tube Od
(m) Cross For 1/4" Tube Id
(n) 3/8" OD x 1/4" ID Tubing
(o) 1/4" OD x 0.170" ID Tubing
(p) 1/8" OD x 1/16" ID Tubing
(q) 3/4" OD x 5/8" ID Tubing
(r) Solenoid Valve
(s) Laboratory Stopper
(t) Odor Cup
(u) Vacuum Pump
(v) Air Push Pump
(w) One Way Check Valve
(x) Fixed Base For Holding Tubes
(y) Elevator for Move The Drosophilae
(z) Few Small Size Hole Through The Elevator, The Drosophilae Can't Get In.
(aa) Hole Through The Elevator, The Drosophilae Can Move In
(bb) Sensor is embeded in the tubing

Figure 3. (A) A schematic diagram of the rotating automated *Drosophila* training and testing system. (B) A schematic diagram of the linear automated *Drosophila* training and testing system.

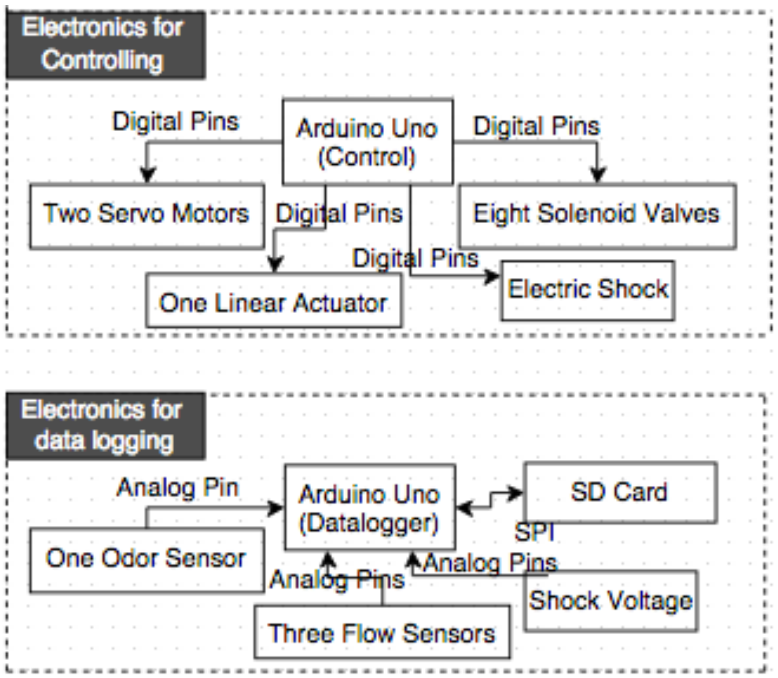


Figure 4. Electronics schematic

The ATMega 328 has a flash memory of 32 KB, of which 0.5 KB are reserved for the bootloader. 2 KB of SRAM, 1 KB of EEPROM, and a clock speed of 16 MHz. The Arduino Mega 328 can be powered through a USB connection or with an external power supply.

Additionally, the Arduino platform was particularly attractive because of its open-source physical computing platform, based on a simple I/O board, and a development environment that implements the Processing, the LabVIEW and language, seen below in Figure 5.

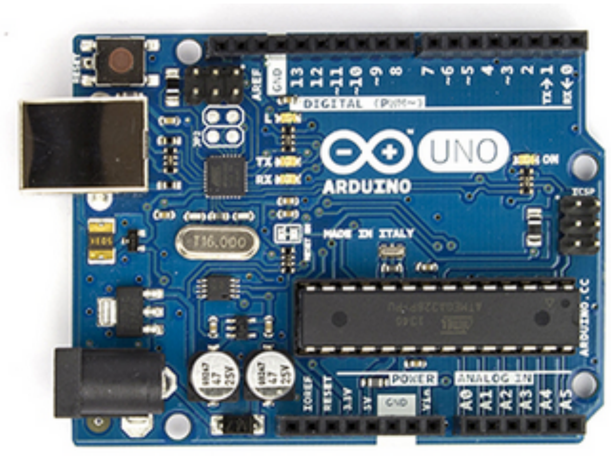


Figure 5. Arduino Mega 328 Microcontroller Board

The ATMega 328 comes with an Atmel AVR core that combines a wide instruction set with 32 general purpose working registers. All 32 registers are directly coupled to the ALU (Arithmetic Logic Unit), allowing two independent registers to be accessed in one single execution of the instruction per clock cycle. In order to maximize parallelism, the AVR uses a Harvard architecture, with separate memories and buses for program and data. Figure 6 shows the block diagram of the AVR architecture.

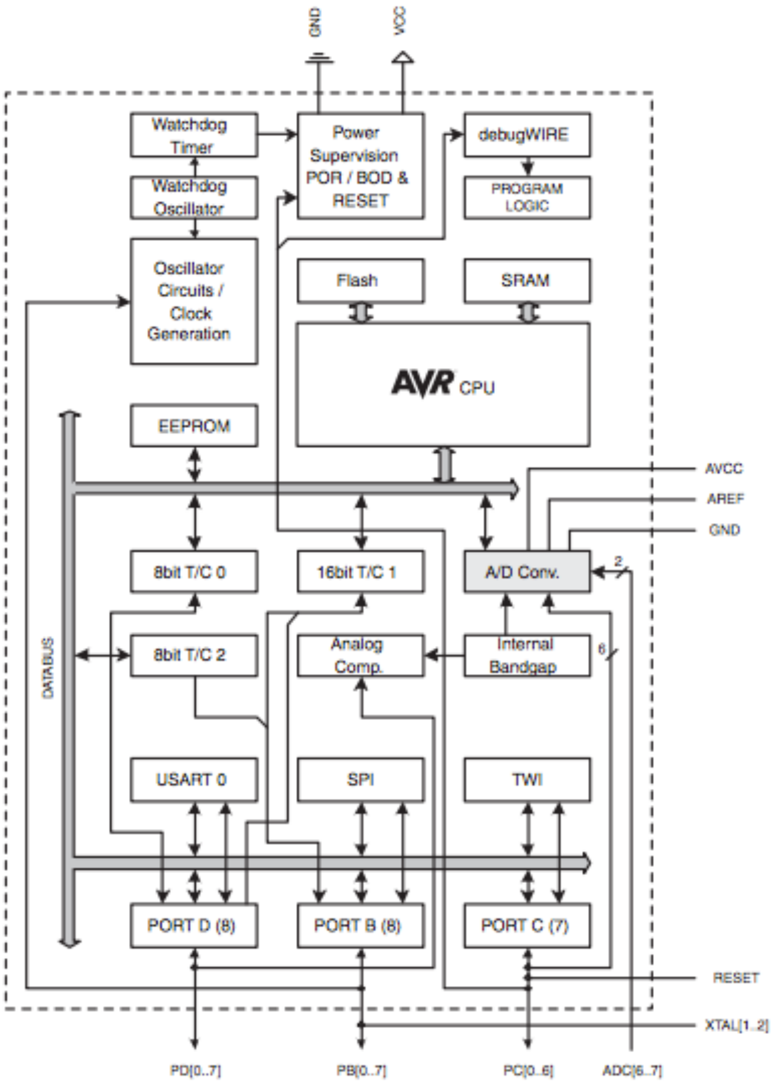


Figure 6. Arduino Mega 328 Microcontroller Block Diagram

Solenoid Valve

SMC VDW31-6G-2-02N Solenoid Valve [6]

The solenoid valve chosen for this project was the SMC VDW31-6G-2-02N, shown in Figure 7. Few considerations had to be taken into account when making the appropriate selection. Ideally the solenoid valve should have the following features:

- Type (e.g. 2-way, 3-way, or 4-way) and operation (e.g. normally open, normally closed, or universal) of the valve
- The media that flow through the valve and the media temperature
- Size of the orifice in the valve
- The maximum differential pressure applied into the inlet port
- Environment of a solenoid valve

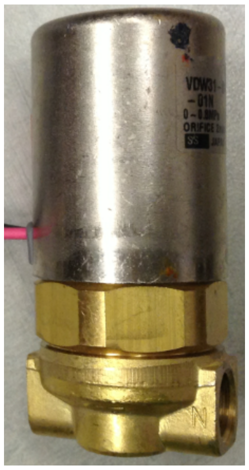
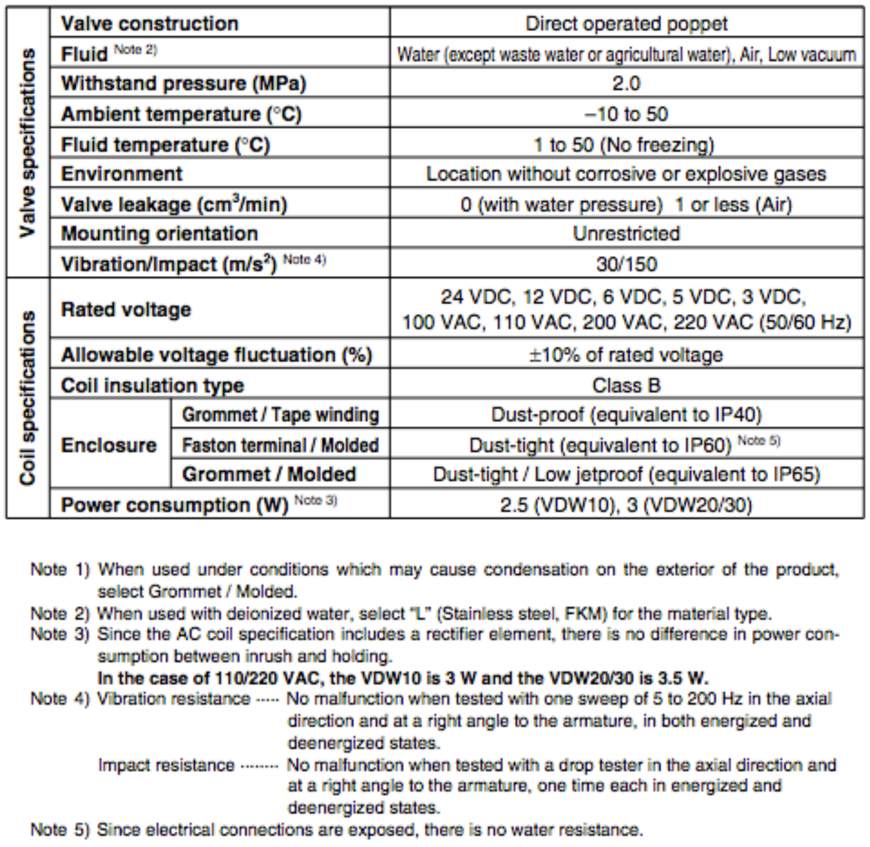


Figure 7. SMC VDW31-6G-2-02N Solenoid Valve

We decided to use the 2-way and normally closed solenoid valve. In our experimental model, everything passing through the solenoid valve is humid with a temperature of 26 degrees Celsius. The environment of the valve is 80%-85% humidity and 26 degrees Celsius temperature.

The SMC VDW31-6G-2-02N solenoid valve offered several outstanding performance features that fulfilled our needs. Table 1 is SMC VDW30 series standard specifications. Additionally, the use of a unique magnetic material reduces the operating resistance of moving parts, while improving service life, wear and corrosion resistance. Changing of the coil is made easy by means of the clip design.

Table 1. SMC VDW30 series standard specifications



Valve specifications	**Valve construction**		Direct operated poppet
	Fluid Note 2)		Water (except waste water or agricultural water), Air, Low vacuum
	Withstand pressure (MPa)		2.0
	Ambient temperature (°C)		–10 to 50
	Fluid temperature (°C)		1 to 50 (No freezing)
	Environment		Location without corrosive or explosive gases
	Valve leakage (cm^3/min)		0 (with water pressure) 1 or less (Air)
	Mounting orientation		Unrestricted
	Vibration/Impact (m/s^2) Note 4)		30/150
Coil specifications	**Rated voltage**		24 VDC, 12 VDC, 6 VDC, 5 VDC, 3 VDC, 100 VAC, 110 VAC, 200 VAC, 220 VAC (50/60 Hz)
	Allowable voltage fluctuation (%)		±10% of rated voltage
	Coil insulation type		Class B
	Enclosure	**Grommet / Tape winding**	Dust-proof (equivalent to IP40)
		Faston terminal / Molded	Dust-tight (equivalent to IP60) Note 5)
		Grommet / Molded	Dust-tight / Low jetproof (equivalent to IP65)
	Power consumption (W) Note 3)		2.5 (VDW10), 3 (VDW20/30)

Note 1) When used under conditions which may cause condensation on the exterior of the product, select Grommet / Molded.
Note 2) When used with deionized water, select "L" (Stainless steel, FKM) for the material type.
Note 3) Since the AC coil specification includes a rectifier element, there is no difference in power consumption between inrush and holding.
In the case of 110/220 VAC, the VDW10 is 3 W and the VDW20/30 is 3.5 W.
Note 4) Vibration resistance ····· No malfunction when tested with one sweep of 5 to 200 Hz in the axial direction and at a right angle to the armature, in both energized and deenergized states.
Impact resistance ········ No malfunction when tested with a drop tester in the axial direction and at a right angle to the armature, one time each in energized and deenergized states.
Note 5) Since electrical connections are exposed, there is no water resistance.

Solenoid Valve Control Circuit [7]

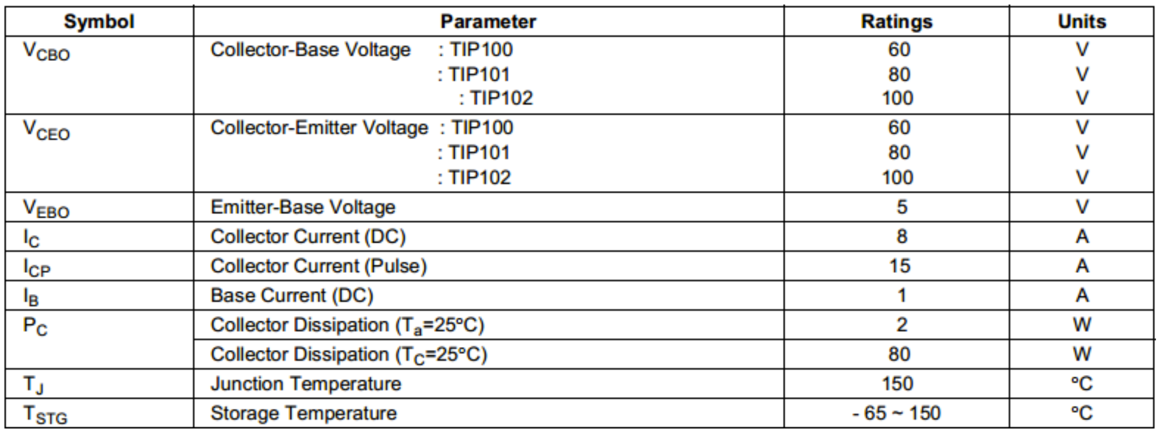
A solenoid valve is an electromechanically operated valve. An electric current through a solenoid controls the valve. SMC VDW31-6G-2-02N is a 12V solenoid valve and consumes 3W of power. We can calculate the current that flows through the coil of the solenoid valve using following equation:

$$I = P / V \Rightarrow I = 3\text{W} / 12\text{V} \Rightarrow I = 0.25\text{A} \qquad (1)$$

Arduino Uno is not capable of providing enough power for such loads, in terms of both voltage and current. The solution was a transistor driver. The purpose of this transistor driver is to amplify the Arduino Uno current or voltage (or both) and provide enough power for the solenoid valve. The transistor model being used is the TIP 102 BJT. Table 2 and Table 3 [8] shows two parts of the data sheet for this transistor.

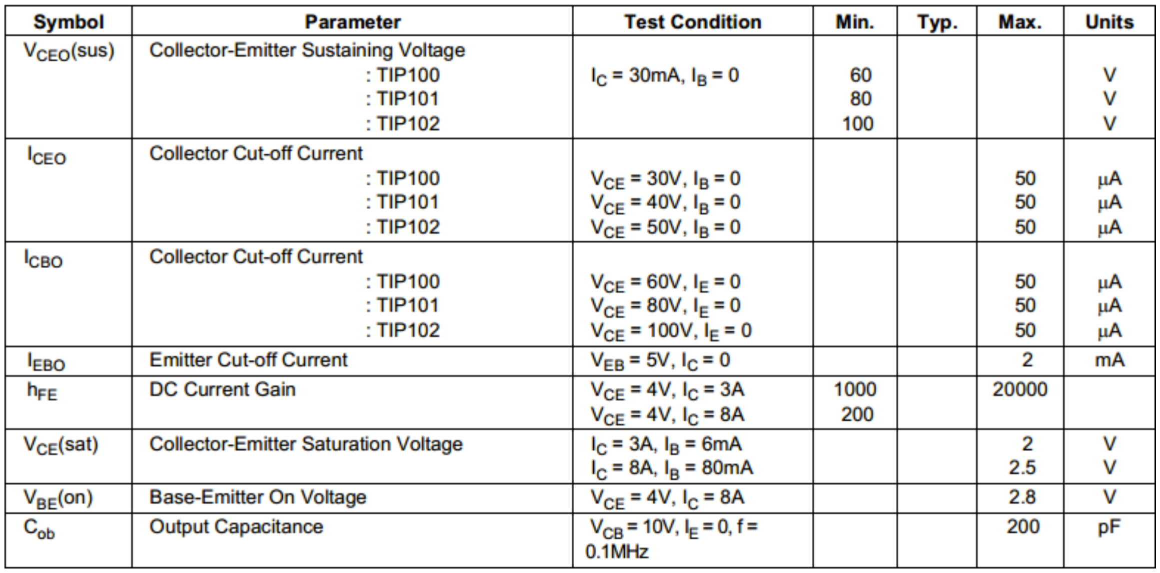
The transistor model was chosen for several reasons. First, we need to provide 0.25A collector current to the solenoid valve. An additional 20% collector current is a safe number, so we need our transistor to be able to provide about 0.3A.

Table 2. First part of the Data Sheet for the TIP 102 Bipolar Transistor



Symbol	Parameter	Ratings	Units
V_{CBO}	Collector-Base Voltage : TIP100 : TIP101 : TIP102	60 80 100	V V V
V_{CEO}	Collector-Emitter Voltage : TIP100 : TIP101 : TIP102	60 80 100	V V V
V_{EBO}	Emitter-Base Voltage	5	V
I_C	Collector Current (DC)	8	A
I_{CP}	Collector Current (Pulse)	15	A
I_B	Base Current (DC)	1	A
P_C	Collector Dissipation (T_a=25°C)	2	W
	Collector Dissipation (T_C=25°C)	80	W
T_J	Junction Temperature	150	°C
T_{STG}	Storage Temperature	- 65 ~ 150	°C

Table 3. Second part of the Data Sheet for the TIP 102 Bipolar Transistor



Symbol	Parameter	Test Condition	Min.	Typ.	Max.	Units
V_{CEO}(sus)	Collector-Emitter Sustaining Voltage : TIP100 : TIP101 : TIP102	I_C = 30mA, I_B = 0	60 80 100			V V V
I_{CEO}	Collector Cut-off Current : TIP100 : TIP101 : TIP102	V_{CE} = 30V, I_B = 0 V_{CE} = 40V, I_B = 0 V_{CE} = 50V, I_B = 0			50 50 50	μA μA μA
I_{CBO}	Collector Cut-off Current : TIP100 : TIP101 : TIP102	V_{CE} = 60V, I_E = 0 V_{CE} = 80V, I_E = 0 V_{CE} = 100V, I_E = 0			50 50 50	μA μA μA
I_{EBO}	Emitter Cut-off Current	V_{EB} = 5V, I_C = 0			2	mA
h_{FE}	DC Current Gain	V_{CE} = 4V, I_C = 3A V_{CE} = 4V, I_C = 8A	1000 200		20000	
V_{CE}(sat)	Collector-Emitter Saturation Voltage	I_C = 3A, I_B = 6mA I_C = 8A, I_B = 80mA			2 2.5	V V
V_{BE}(on)	Base-Emitter On Voltage	V_{CE} = 4V, I_C = 8A			2.8	V
C_{ob}	Output Capacitance	V_{CB} = 10V, I_E = 0, f = 0.1MHz			200	pF

The TIP 102 is able to provide up to 8A (I_C), which is more than enough. Second, for the voltage, we need to power the transistor with 12V. Again, overpower of this valve by 20% should provide about 14.4V to the transistor. The TIP102 can switch up to 100V (V_{CEO}), which is again more than enough.

The TIP 102 transistor is connected to activate the solenoid valve, as shown in Figure 8. A prototype photo of the solenoid valve control circuit is shown in Figure 9. We provide an external power supply of 12V. The purpose of the diode is to prevent reverse electric and magnetic fields from damaging the transistor (reverse electric and magnetic fields is the voltage produced when current through a coil is switched off).

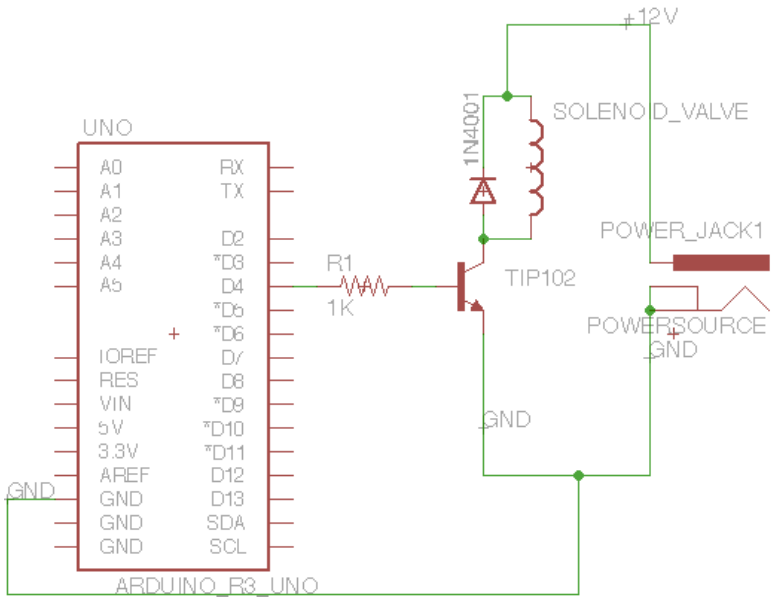


Figure 8. Driving a Solenoid With a Transistor Schematic Diagram

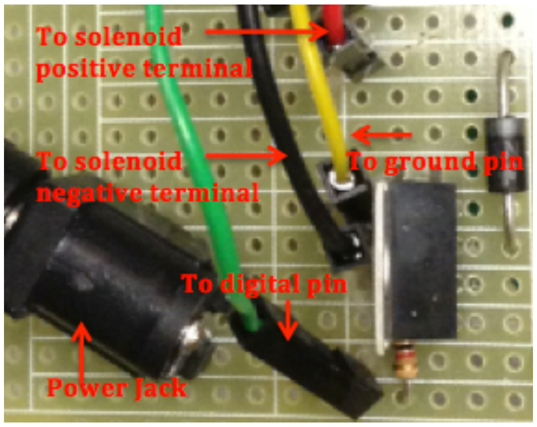


Figure 9. Solenoid Valve Control Circuit Prototype Photo

Servo Motor

FS5106B Servo Motor [9]

A servo is a device that can rotate to an arbitrary position, as set by the user. The servo usually consists of a small DC (direct current) electric motor, several gears and a head where an arm or wheel can be attached. When the user tells the servo what angular position to move to, the servo rotates and holds that position until further input is specified.

In our RATTS design, the 3D printed parts include the cylinder, the screen filter and the non-rotating part (see chapter II). One servomotor controls the cylinder and the other servomotor controls the screen filter. The servomotors need to provide the required torque, speed and accuracy for the RATTS system to perform as designed. The FS5106B Servomotor seemed to fit these requirements (Figure 10).

Figure 10. FS5106B Servo Motor

The FS5106B Servomotor offered the following characteristics seen in Table 4:

Table 4. FS5106B Servo Motor Specifications Table

Power	4.8V	6V
Speed	0.18sec/60degree	0.16sec/60degree
Torque	5.0kg.cm/69.56oz.in	6.0kg.cm/83.47oz.in
Weight	40g(1.41oz)	
Size	40.8*20.1*38.0mm	
Application	Air plane Car boat	

Servo Motor Control Circuit

Three wires control the servo motors: two to provide the DC power that the motor needs, and one that sends the signal, controlling the servo. The signal wire works by sending the servo a series of pulses, which are interpreted by internal circuitry. By varying the timing of each pulse, the servo can determine exactly which position to move to. A schematic diagram of the driving FS5106B Servomotor using Arduino Uno is shown in Figure 11.

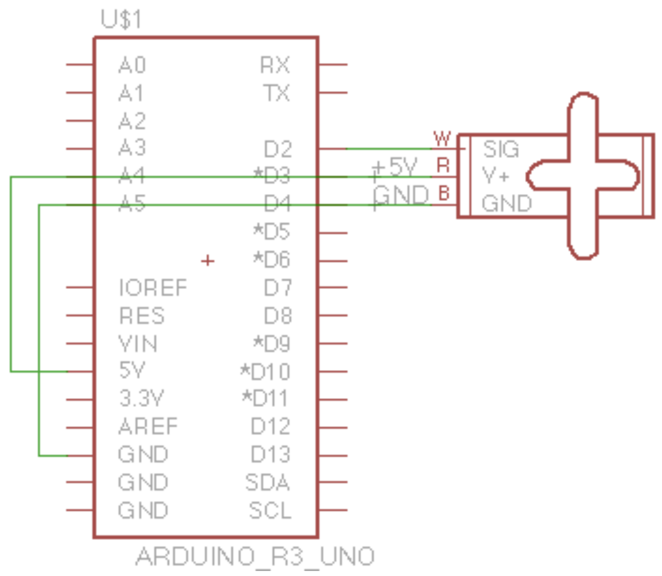


Figure 11. Driving Servo Motor Schematic Diagram

Linear Actuator

LACT6P Linear Actuator [10]

A linear actuator is a mechanism, or assembly, that creates motion and force along a straight line utilizing an externally applied energy source. The linear actuator chosen for LATTS design was the LACT6P by Creative Werks Inc. (Figure 12). LACT6P has gear-driven drive motors that extend the shaft with 110 lbs. of force and 550 lbs. of static load rating. With built-in limit switches, ACME dive system, IP 63 dust and water resistant rating. Also, a potentiometer allows us to preset three different stop locations along the actuator's line of travel using the controller we have available. This 6'' actuator has a retracted length of 11.49 inches and an extended length of 17.49 inches. At max load the actuator travels 0.50 inches per second. Input voltage is 12VDC and recommended fuse is 10 Amp. This specification is observed in Table 5.

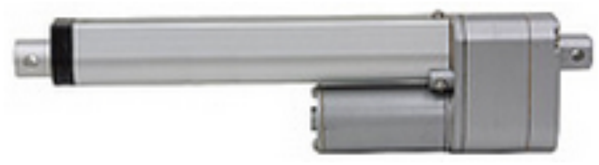
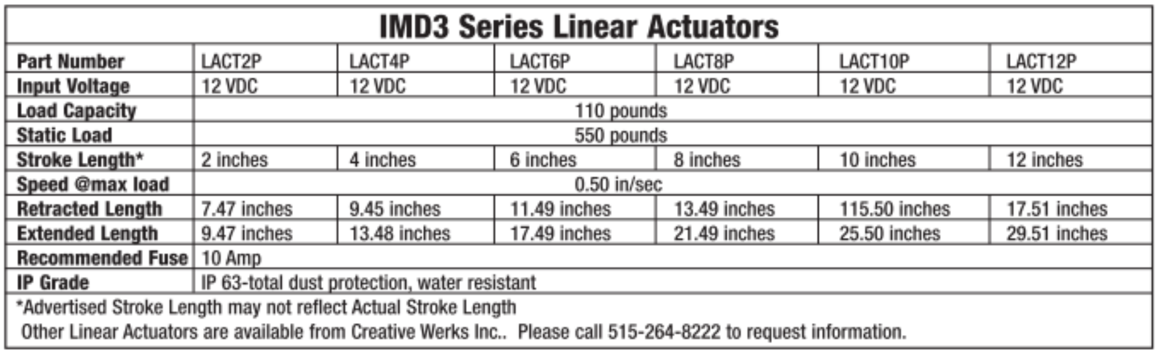


Figure 12. LACT6P Linear Actuator

Table 5. LACT6P Linear Actuator Specifications Table



IMD3 Series Linear Actuators

Part Number	LACT2P	LACT4P	LACT6P	LACT8P	LACT10P	LACT12P
Input Voltage	12 VDC	12 VDC	12 VDC	12 VDC	12 VDC	12 VDC
Load Capacity	110 pounds					
Static Load	550 pounds					
Stroke Length*	2 inches	4 inches	6 inches	8 inches	10 inches	12 inches
Speed @max load	0.50 in/sec					
Retracted Length	7.47 inches	9.45 inches	11.49 inches	13.49 inches	115.50 inches	17.51 inches
Extended Length	9.47 inches	13.48 inches	17.49 inches	21.49 inches	25.50 inches	29.51 inches
Recommended Fuse	10 Amp					
IP Grade	IP 63-total dust protection, water resistant					

*Advertised Stroke Length may not reflect Actual Stroke Length
Other Linear Actuators are available from Creative Werks Inc.. Please call 515-264-8222 to request information.

Linear Actuator Control Circuit [11]

The LACT6P Linear Actuator with feedback, and the cables with unterminated wire ends, is shown in Figure 13. White, yellow and blue mark the three potentiometer leads. Red and black mark two power leads. We use two relays to control the LACT6P Linear Actuator's extension and retraction. The relays chosen for controlling linear actuator was the RobotGeek Relay (Figure 14). The schematic diagram of the RobotGeek Relay is shown in Figure 15 [12].

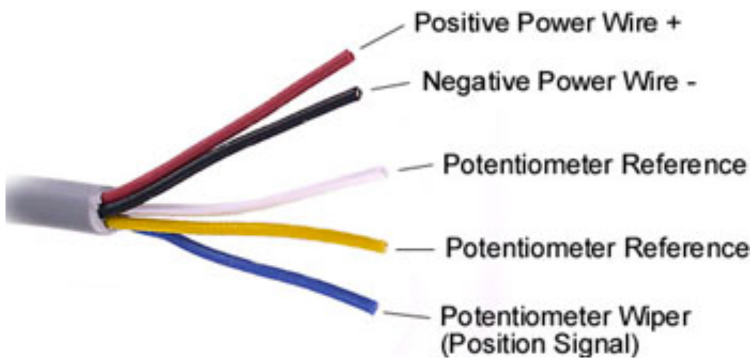


Figure 13. Unterminated Wire Ends for LACT6P Linear Actuator

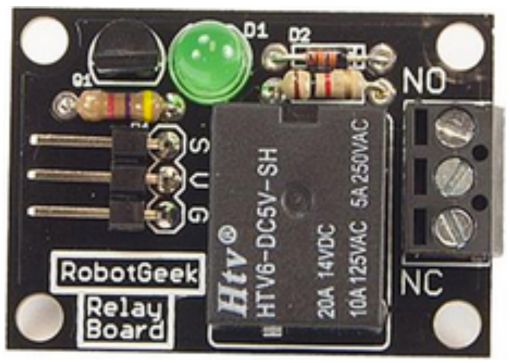


Figure 14. RobotGeek Relay

The schematic diagram and prototype photo for controlling LACT6P Linear Actuator by using Arduino Uno are shown in the Figure 16 and Figure 17, respectively. We provide an external power supply of 12V. The external power supply VDD is connected to two relays NO (Normally Open) terminal, and the GND is connected to two relay NC (Normally Close) terminals. The two power leads of the linear actuator individually connect to two relay C (Common) terminals. We connect the red lead to relay 2's C terminal, and the black lead to relay 1's C terminal. The three potentiometer leads are connected to the Arduino Uno's pins. The white, yellow and blue leads separately connect to Arduino Uno's board GND, 5V and A1 pins. All these wire connections are closely related to the software development (see chapter IV).

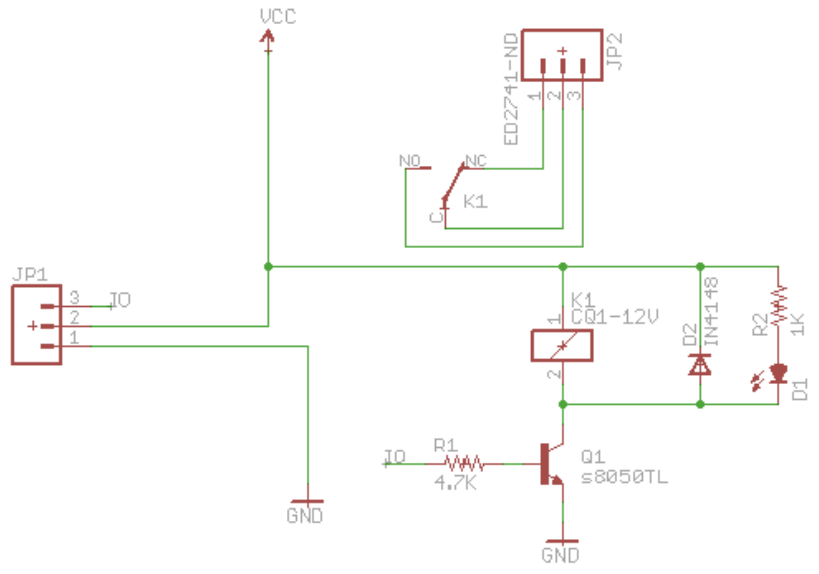


Figure 15. RobotGeek Relay Schematic Diagram

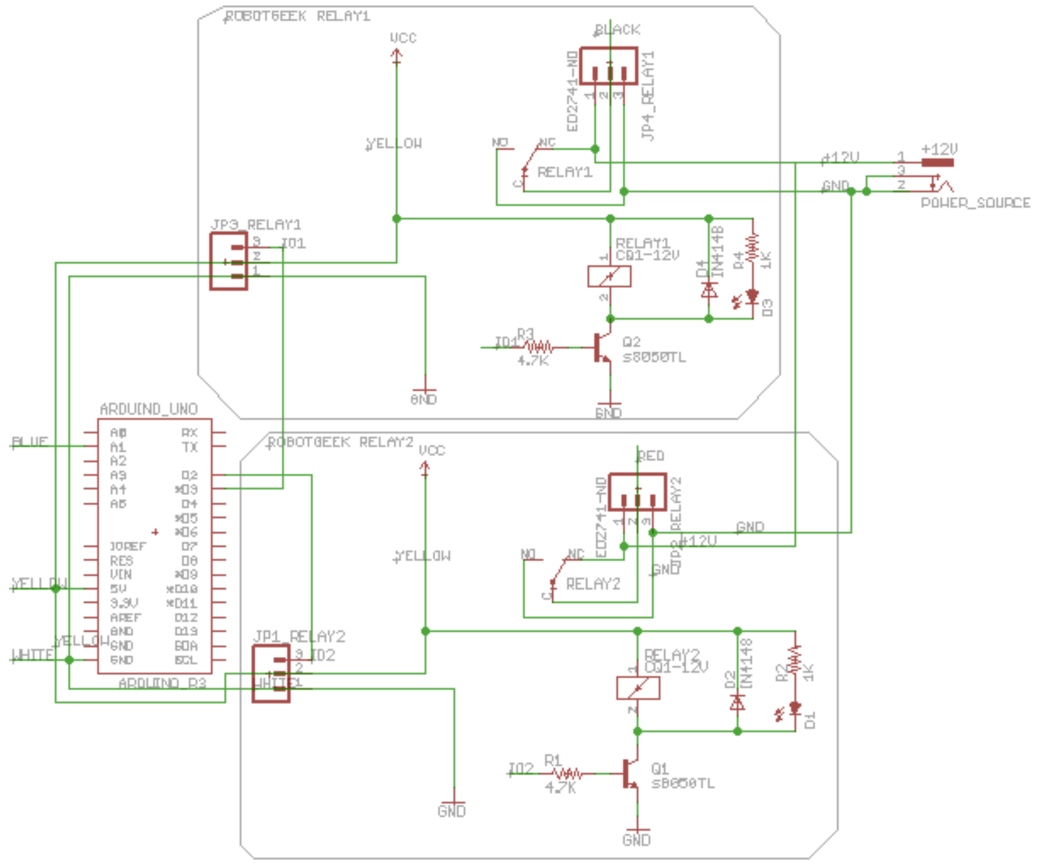


Figure 16. Controlling Linear Actuator Schematic Diagram

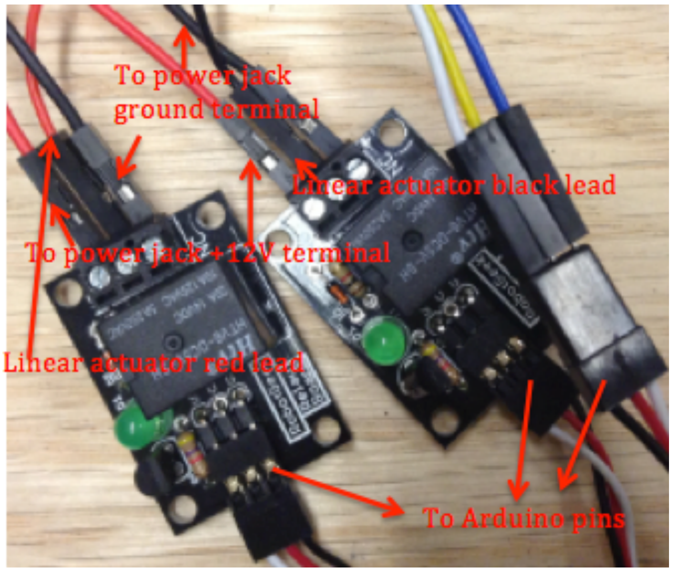


Figure 17. Controlling Linear Actuator Prototype Photo

Electric Shock

SD9 Square Pulse Stimulator

Based on the model described in the chapter I, we need to provide 80V electric shock to the *Drosophila* when required. A SD9 square pulse stimulator[13] delivers the current 80V (see Figure 18).

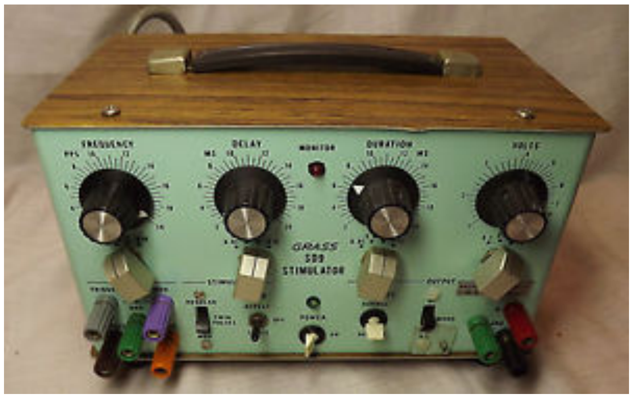


Figure 18. SD9 Square Pulse Stimulator

Electric Shock Control Circuit

We needed an electric shock control circuit to control the 80V shock time. The schematic diagram of the electric shock control circuit is shown in Figure 19. We used the RobotGeek relay (see Figure 14). The jumper

JP4_RELAY3 pin 1 is connected to the ground, pin 2 is connected to the training tube where the *Drosophila* are located during the training section, and pin 3 is connected to the positive 80V provided by the SD9 square pulse stimulator. The training tube also connects with the ground (see Figure 39). When the relay doesn't receive the 5V signal, the training tube connects to two grounds. When the relay does receive the 5V signal, the training tube connects to 80V and the ground. The copper circuit inside of the training tube will provide 80V shocks to the *Drosophila*. The prototype photo of electric shock control circuit is shown in Figure 20.

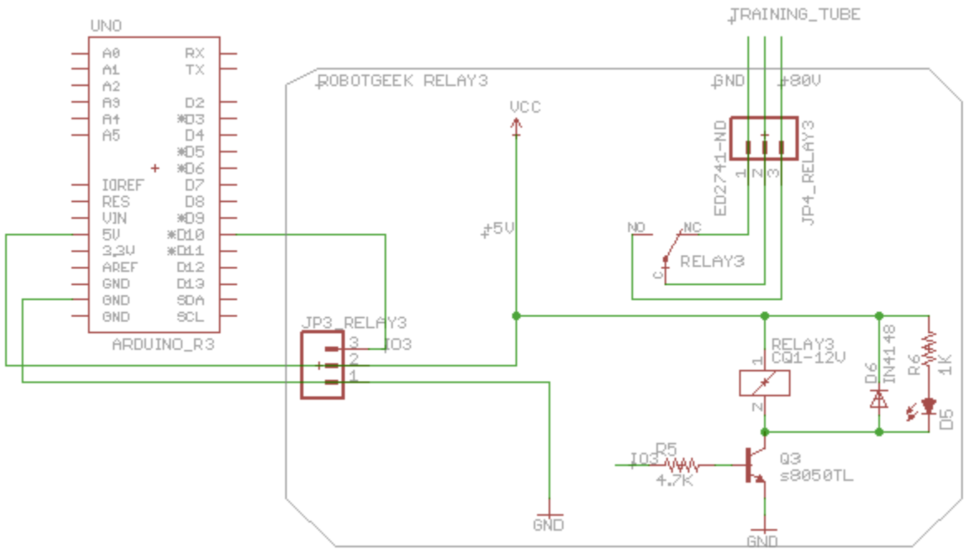


Figure 19. Electric Shock Control Circuit Schematic Diagram

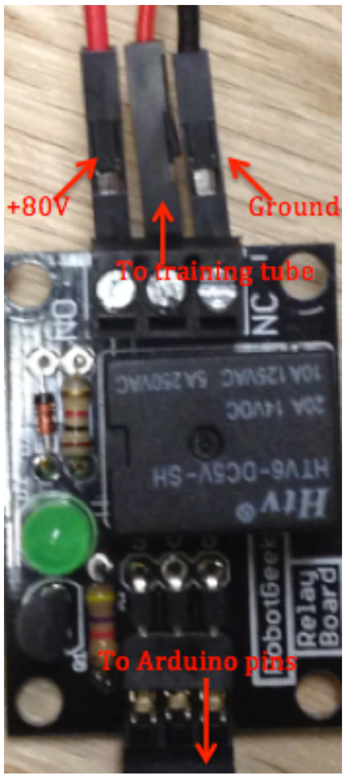


Figure 20. Electric Shock Control Circuit Prototype Photo

<u>Odor Sensor</u>

TGS 2620 – for the detection of Solvent Vapors [14]

We needed to include fail-safe tests to ensure that odors are being delivered correctly. We chose the TGS 2620 odor sensor, which detects the odorants 3-octanol and 4-methylcyclohexanol, the volatile chemicals that are most often used for *Drosophila* olfactory experiments (see chapter I). The Taguchi Gas Sensor (TGS) 2620 as shown in Figure 21 has high sensitivity to the vapors of organic solvents, as well as other volatile vapors. The sensing element is comprised of a metal oxide semiconductor layer formed on an alumina substrate of a sensing chip, together with an integrated heater [14]. In the presence of a detectable gas, the sensor's conductivity increases depending on the gas concentration in the air [14].

Figure 21. TGS 2620 – for the detection of Solvent Vapors

Odor Sensor Data Logging Circuit [15]

The change of conductance in the sensor in the presence of 3-octanol and 4-methylcyclohexanol can be used to measure the concentration of 3-octanol and 4-methylcyclohexanol, as shown in the schematic circuit in Figure 22. A prototype photo of one odor sensor data logging circuit is shown in Figure 23. The electrical properties of the detector require that the sensor be heated via an internal heater.

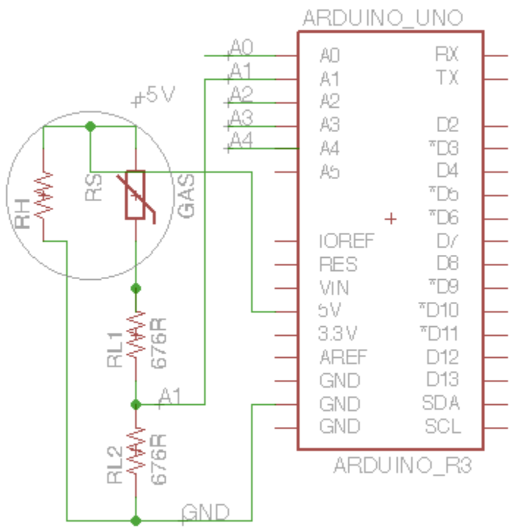


Figure 22. TGS 2620 Odor Sensor Data Logging Schematic Diagram

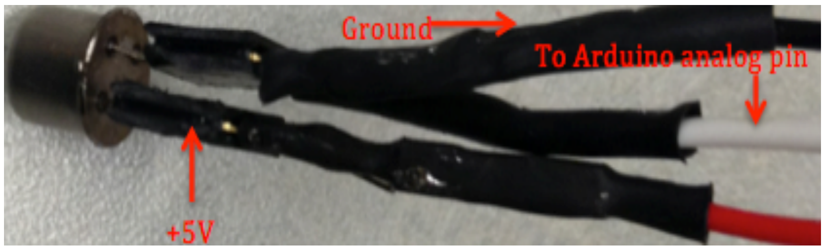


Figure 23. TGS 2620 Odor Sensor Data Logging Prototype Photo

In this configuration, the sensor has a certain resistance R_0 in 300ppm of ethanol. Resistance R_S can be expressed by the power function [16]:

$$R_S = A[C]^{-\alpha} \qquad (2)$$

Where R_S is the actual sensor resistance, A is a coefficient for the gas at concentration $[C]$, and α is the slope of the curve as shown in Figure 24 [14].

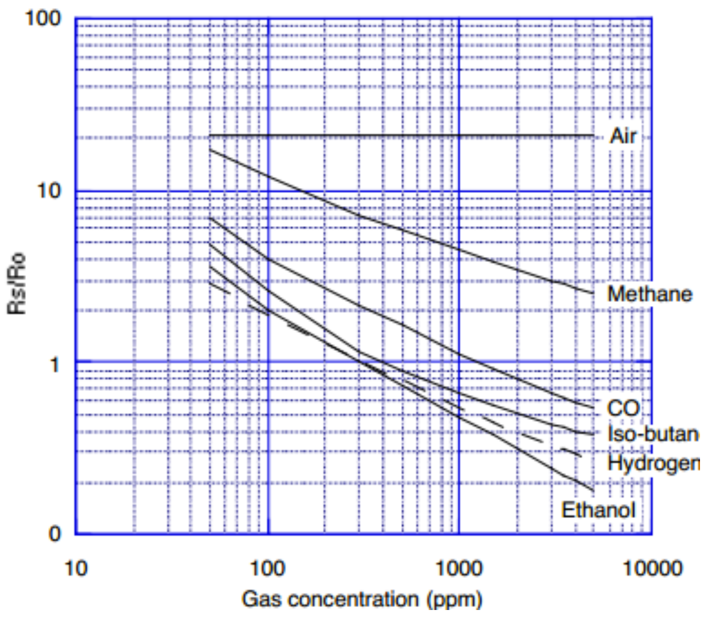


Figure 24. Sensitivity Characteristics of TGS 2620

For the application in this project, we measured R_S in a simple electronic circuit where the voltage drop over a precision resistor R_{L2} in series with R_S and R_{L1} was measured (Figure 22). From such a set-up, the sensor resistance can be determined as:

□□2□□□□−□□1−□□2 (3)

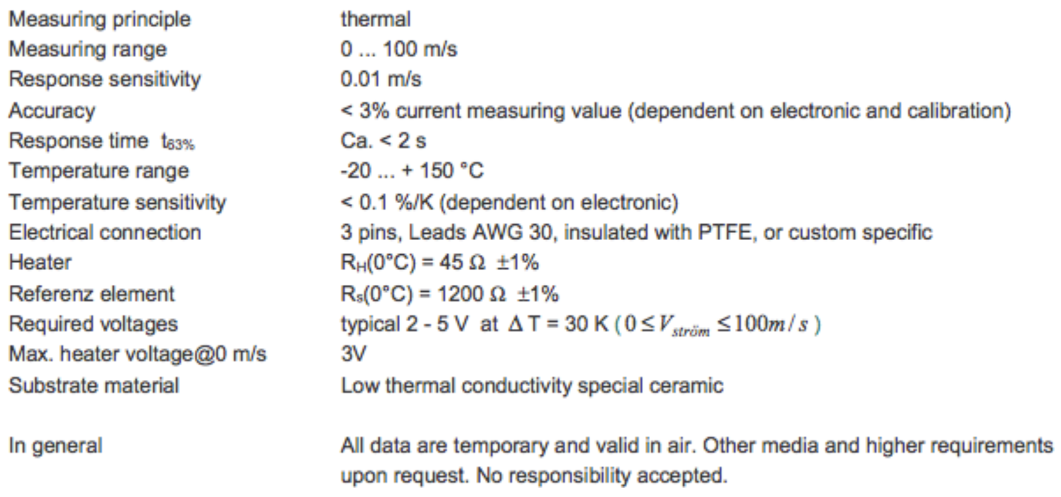
V_C is the supply voltage of 5.0V DC, and V_{OUT} is the voltage measured over the precision resistor R_{L2}. Finally, the ratio between the actual sensor resistance R_S and the 300ppm of ethanol resistance R_0 is the sensor signal of interest to deduce 3-octanol and 4-methylcyclohexanol concentrations (Figure 24).

Flow Sensor

FS5 Thermal Mass Flow Sensor [17]

The FS5 flow sensor (Figure 25) element consists of two temperature dependent platinum-resistors, both deposited on one chip. The low-ohm resistor with a small area is used as heater, whereas the other high-ohm resistor serves to measure the reference temperature. Using a bridge circuit, the differing resistance value of the two elements leads to differential (self) heating. The (self) heating is dependent upon the applied voltage, the mass flow and the media in which the sensor is located. The measuring principle of the sensor can be used for large operation ranges, from 0…0.1m/s up to 100m/s. The FS5 flow sensor offered the following characteristics seen in Figure 26.

Figure 25. FS5 Thermal Mass Flow Sensor



Measuring principle	thermal
Measuring range	0 ... 100 m/s
Response sensitivity	0.01 m/s
Accuracy	< 3% current measuring value (dependent on electronic and calibration)
Response time $t_{63\%}$	Ca. < 2 s
Temperature range	-20 ... + 150 °C
Temperature sensitivity	< 0.1 %/K (dependent on electronic)
Electrical connection	3 pins, Leads AWG 30, insulated with PTFE, or custom specific
Heater	$R_H(0°C) = 45\ \Omega \pm1\%$
Referenz element	$R_s(0°C) = 1200\ \Omega \pm1\%$
Required voltages	typical 2 - 5 V at $\Delta T = 30$ K ($0 \le V_{ström} \le 100 m/s$)
Max. heater voltage@0 m/s	3V
Substrate material	Low thermal conductivity special ceramic
In general	All data are temporary and valid in air. Other media and higher requirements upon request. No responsibility accepted.

Figure 26. Technical Data of FS5 Thermal Mass Flow Sensor

Thermal Flow Sensor Data Logging Circuit [18]

Based on the electronic circuit recommendation of data logging FS5 flow sensor in the FS5 thermal mass flow sensor datasheet (Figure 27), we built the FS5 flow sensor data logging circuit shown in Figure 28. For part 1, we basically used the recommended electronic circuit from the data sheet, using an LM741 differential amplifier [19] and 2N2222 NPN transistor [20] (excluding the calibration resistor). For part 2, we used a 3-terminal positive voltage regulator LM7805 [21], which outputs 5V, and a LT1635 operation amplifier [22] that includes a rail-to-rail output to form voltage subtractor operational amplifier circuits in order to match the input range of the Arduino analog input. A prototype photo of the FS5 thermal mass flow sensor data logging circuit is shown in Figure 29.

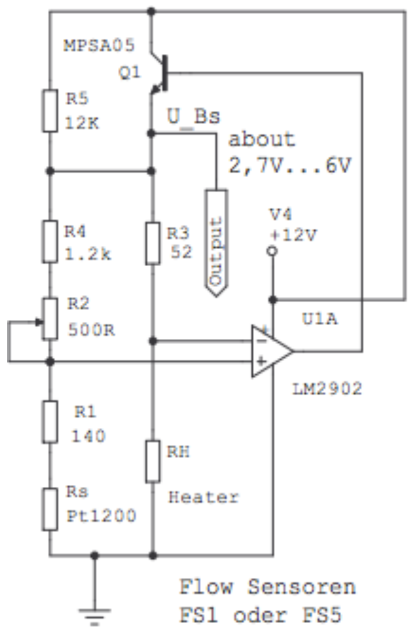


Figure 27. Electronic Circuit Recommendation of a Constant Temperature Anemometer [17]

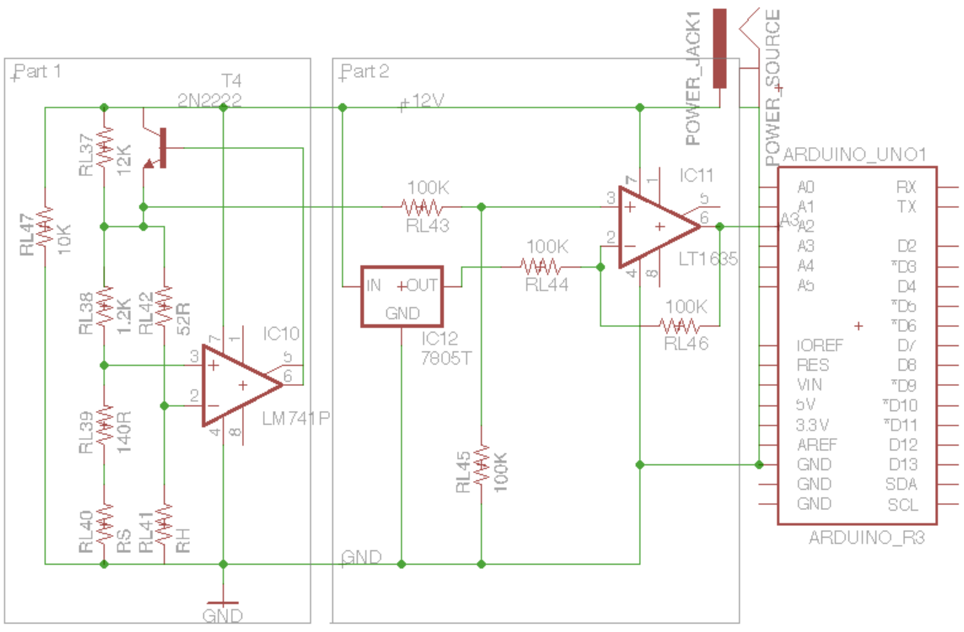


Figure 28. FS5 Thermal Mass Flow Sensor Data Logging Schematic Diagram

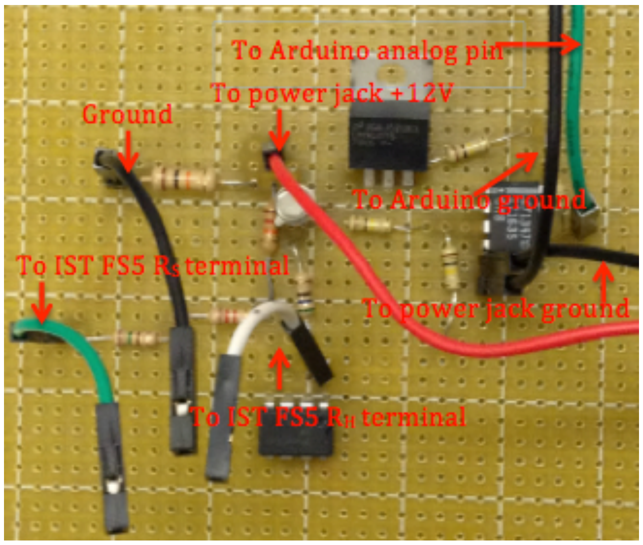


Figure 29. FS5 Thermal Mass Flow Sensor Data Logging Prototype Photo

Voltage Data Logging

We used 80 volts to shock the *Drosophila* whenever the experiment requires. To be able to data log the 80 volts by using Arduino Uno, we needed to build a small voltage divider to reduce the 80 volts into any voltage value between 0V-5V, since the analog input pin voltage of Arduino Uno is limited to 0V-5V. A schematic diagram and prototype photo of the 80V voltage divider are shown in the Figure 30 and Figure 31, respectively.

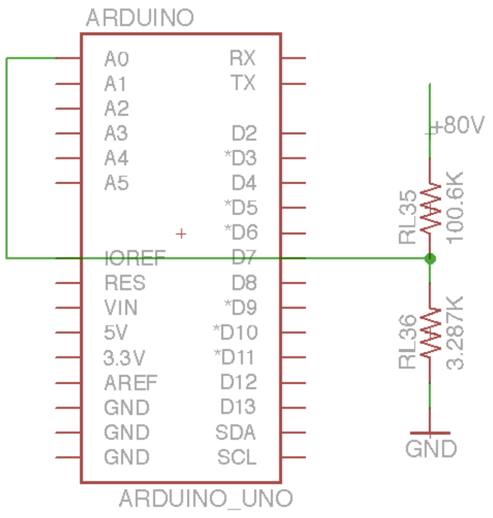


Figure 30. 80V Voltage Divider Schematic Diagram

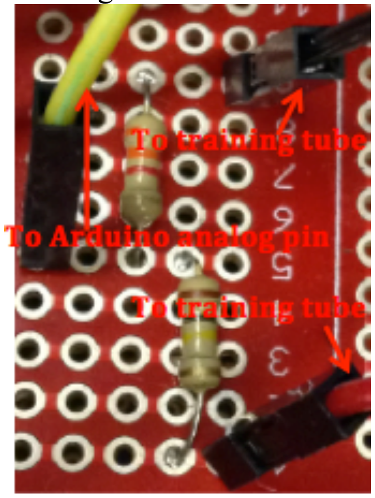


Figure 31. 80V Voltage Divider Prototype Photo

<u>Data Logger Shield</u>

Recording the data of the odor sensor and thermal flow sensors is needed for fail-safe tests to ensure that odors, air flow rates and electrical shocks are be delivered correctly. There are many types of data that need to be recorded, four sensors data output during the entire experimental, and we need to move the entire data to a software program to plot. Therefore, we did not use the Arduino's serial monitor to record the input data. Rather, The Adafruit Data logger shield (Figure 32) was chosen for our project for the following reasons [23]:

- We can quickly save data to files on any FAT16 or FAT32 format SD card
- Can be read by any plotting, spreadsheet or analysis program
- Includes Real Time Clock timestamps on all the data with the current time

- Included libraries and example code for both SD and RTC so we can get going quickly

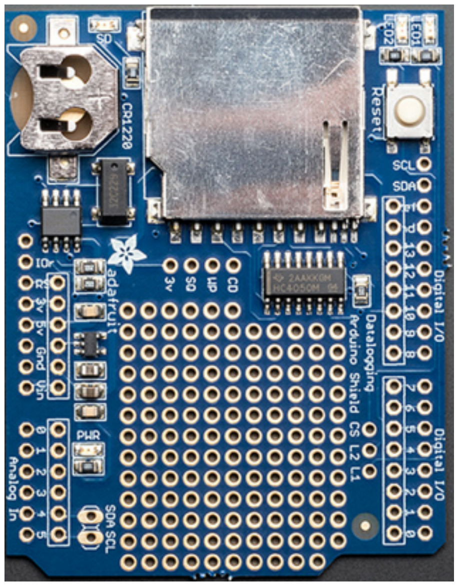


Figure 32. Adafruit Arduino Data Logger Shield

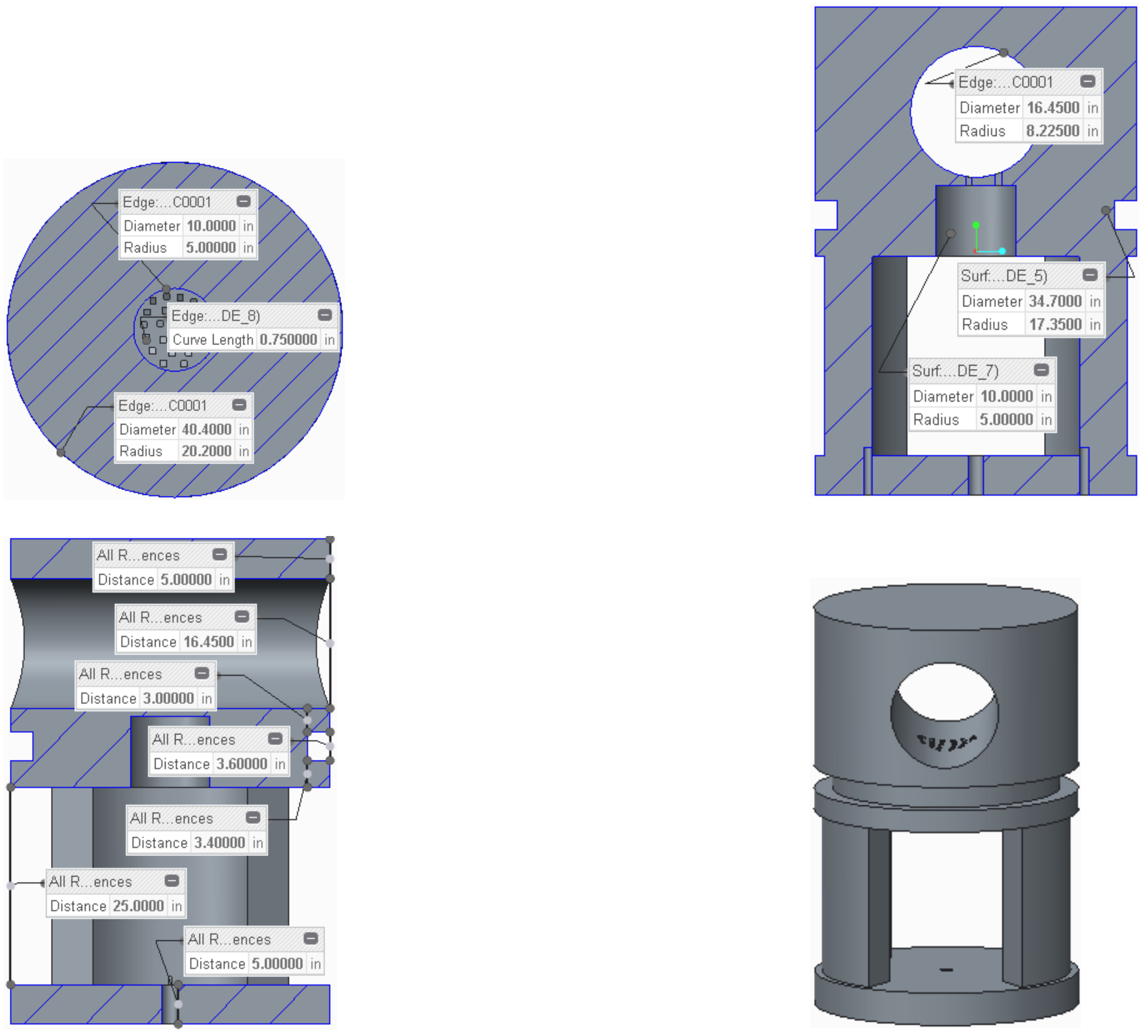
Mechanical Components

RATTS 3D Print Parts

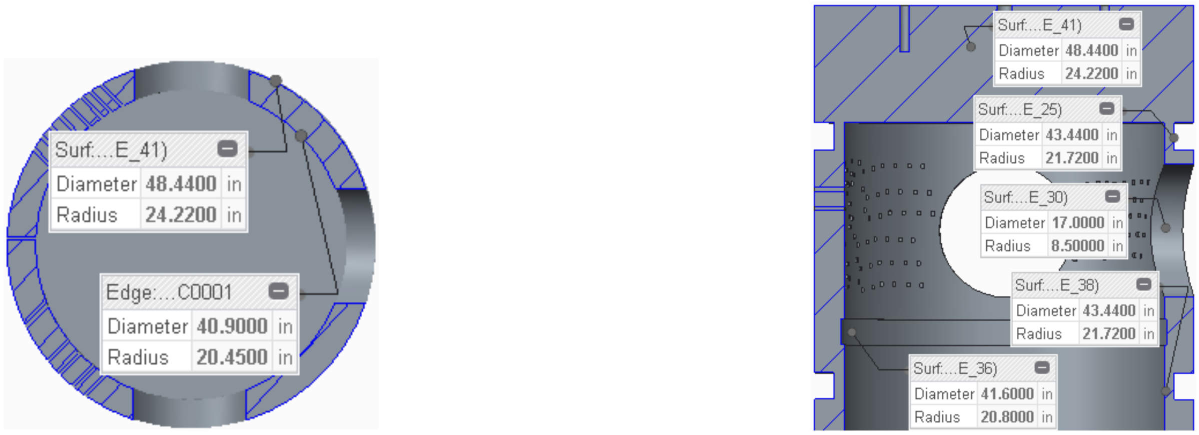
RATTS 3D printed parts mainly include the cylinder, the screen filter and non-rotating component. The 3D printer we used to print the 3D parts is a Stratasys Dimension 768 SST [24]. The 3D printing materials are acrylonitrile butadiene styrene (ABS). PTC creo [25] was chosen by us to design the 3D parts. Figure 33 shows the cross section of the bottom, front and left side views of the designed four 3D parts of RATTS, plus each part's basic view.

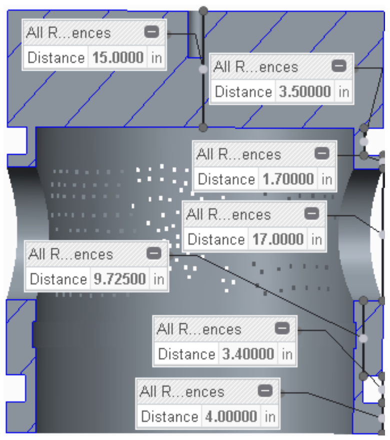
There are slots designed for an O-ring in order to make the whole 3D parts design better sealed to each other (see Figure 33A and B). The air leakage may make the vacuum pump's intensity of suction insufficient, and render the odor concentration to dilute. In Figure 33A, the 25mm length is for the fitting space, which is used for connecting the vacuum pump to the cylinder (see also Figure 34). The 50.175mm length and the 63.525mm length shown in Figure 33C are for the servomotor's space. Figure 34 shows the assembled real 3D parts for RATTS.

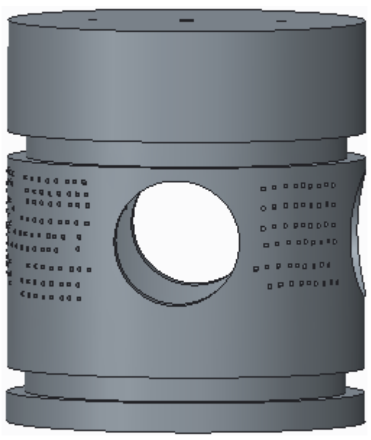
(A)


Edge:...C0001
Diameter 16.4500 in
Radius 8.22500 in
Edge:...C0001
Diameter 10.0000 in
Radius 5.00000 in
Edge:...DE_8)
Curve Length 0.750000 in
Edge:...C0001
Diameter 40.4000 in
Radius 20.2000 in
Surf....DE_5)
Diameter 34.7000 in
Radius 17.3500 in
Surf....DE_7)
Diameter 10.0000 in
Radius 5.00000 in
All R...ences
Distance 5.00000 in
All R...ences
Distance 16.4500 in
All R...ences
Distance 3.00000 in
All R...ences
Distance 3.60000 in
All R...ences
Distance 3.40000 in
All R...ences
Distance 25.0000 in
All R...ences
Distance 5.00000 in

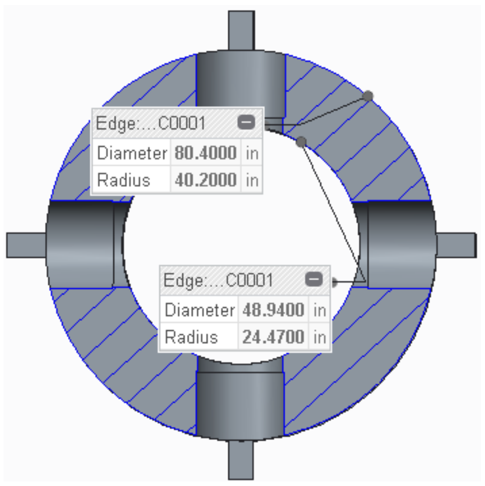
(B)

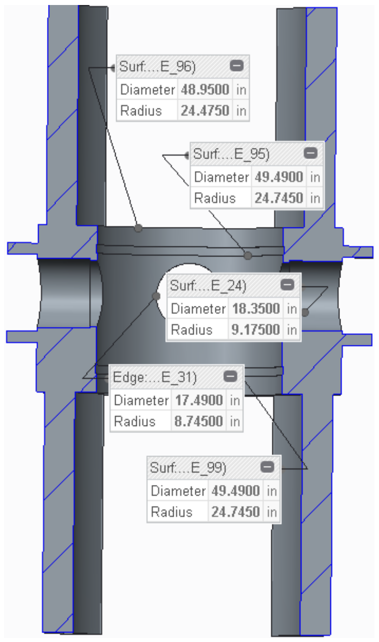

Surf....E_41)
Diameter 48.4400 in
Radius 24.2200 in
Edge:...C0001
Diameter 40.9000 in
Radius 20.4500 in
Surf....E_41)
Diameter 48.4400 in
Radius 24.2200 in
Surf....E_25)
Diameter 43.4400 in
Radius 21.7200 in
Surf....E_30)
Diameter 17.0000 in
Radius 8.50000 in
Surf....E_38)
Diameter 43.4400 in
Radius 21.7200 in
Surf....E_36)
Diameter 41.6000 in
Radius 20.8000 in

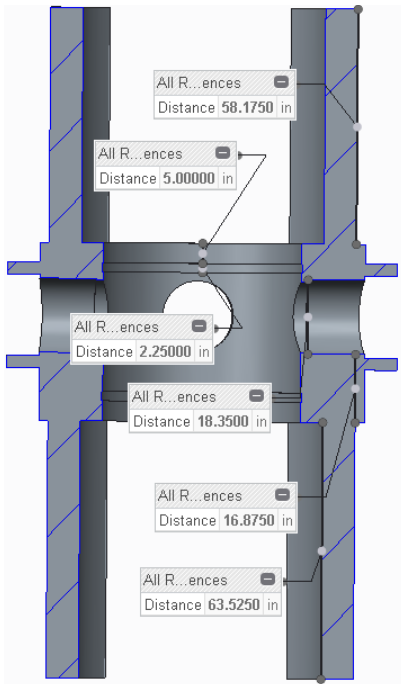

All R...ences
Distance 15.0000 in
All R...ences
Distance 3.50000 in
All R...ences
Distance 1.70000 in
All R...ences
Distance 17.0000 in
All R...ences
Distance 9.72500 in
All R...ences
Distance 3.40000 in
All R...ences
Distance 4.00000 in

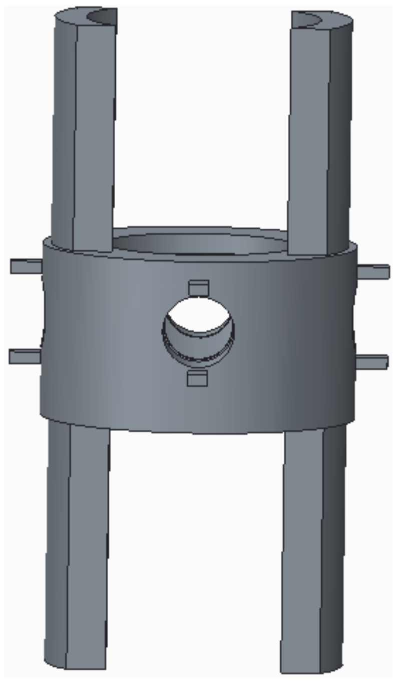


(C)

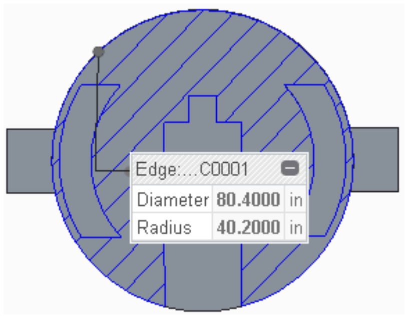

Edge:...C0001
Diameter 80.4000 in
Radius 40.2000 in
Edge:...C0001
Diameter 48.9400 in
Radius 24.4700 in

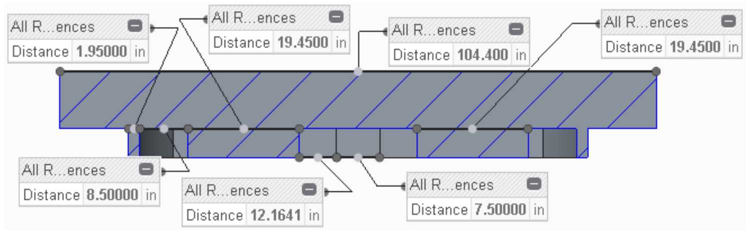

Surf....E_96)
Diameter 48.9500 in
Radius 24.4750 in
Surf....E_95)
Diameter 49.4900 in
Radius 24.7450 in
Surf....E_24)
Diameter 18.3500 in
Radius 9.17500 in
Edge:...E_31)
Diameter 17.4900 in
Radius 8.74500 in
Surf....E_99)
Diameter 49.4900 in
Radius 24.7450 in

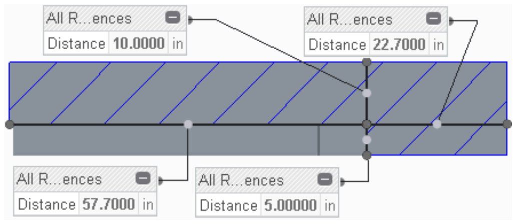

All R...ences Distance 58.1750 in
All R...ences Distance 5.00000 in
All R...ences Distance 2.25000 in
All R...ences Distance 18.3500 in
All R...ences Distance 16.8750 in
All R...ences Distance 63.5250 in



(D)




All R...ences Distance 1.95000 in
All R...ences Distance 19.4500 in
All R...ences Distance 104.400 in
All R...ences Distance 19.4500 in
All R...ences Distance 8.50000 in
All R...ences Distance 12.1641 in
All R...ences Distance 7.50000 in


All R...ences Distance 10.0000 in
All R...ences Distance 22.7000 in
All R...ences Distance 57.7000 in
All R...ences Distance 5.00000 in

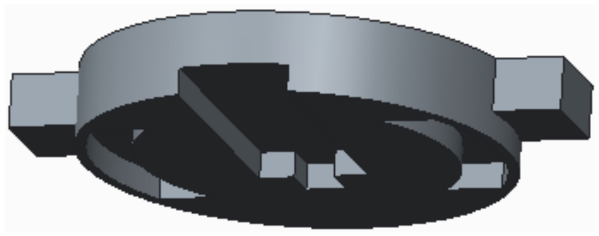


Figure 33. Bottom, front, and left side cross section view of RATTS four 3D parts. We selected the interpret dimensions options in PTC creo, which means 1'' becomes 1mm in the design. (A) RATTS cylinder part's bottom, front, and left side cross section view according to the order left to right and top to bottom. (B) RATTS screen filter part's bottom, front, and left side cross-section view according to the order from left to right and from top to bottom. (C) RATTS non-rotating part's bottom, front, and left side view according to the order from left to right and from top to bottom. (D) RATTS cap part's bottom, front, and left side cross-section view according to the order from left to right and from top to bottom. The cap is for fixing the two servomotors, which are used to rotate the cylinder part and the screen filter part.

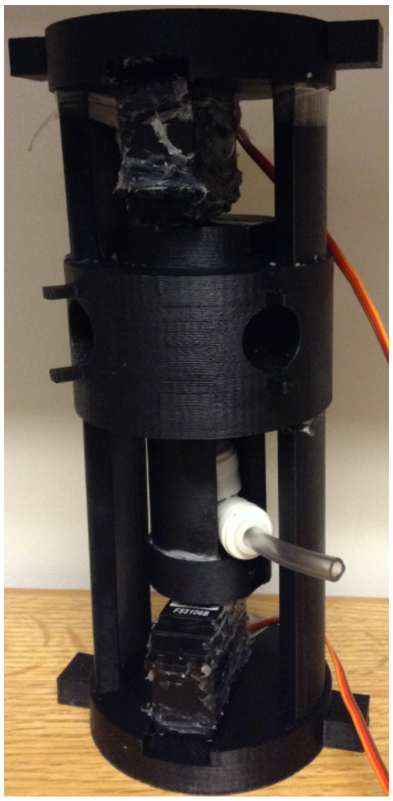
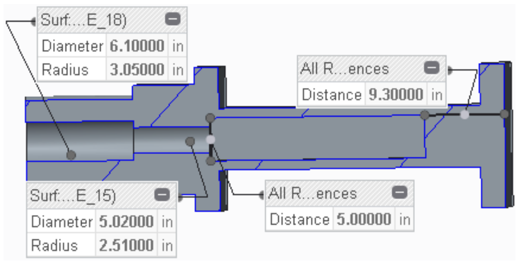


Figure 34. Assembled of the Real 3D Parts Object of RATTS

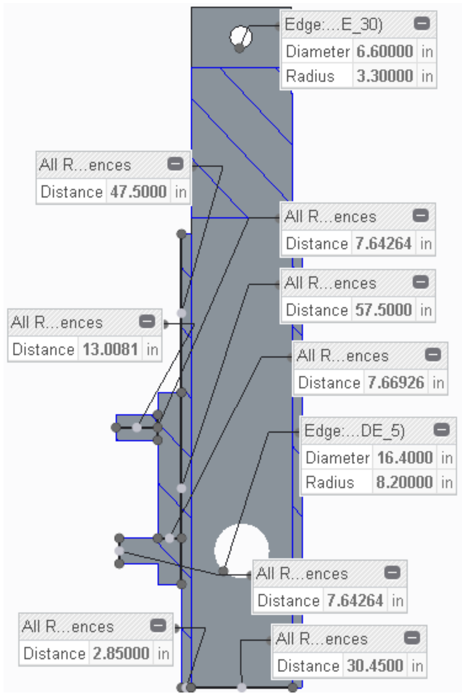
LATTS 3D Print Parts

The LATTS 3D printed parts mainly include the base and elevator. Figure 35 shows the bottom, front and left side cross section views of the designed three 3D LATTS parts. The arrow pointing to the hole as in Figure 35A, will be covered by the small sized mesh, which the *Drosophila* can't pass through. The assembled real 3D parts object of LATTS are shown in Figure 36.

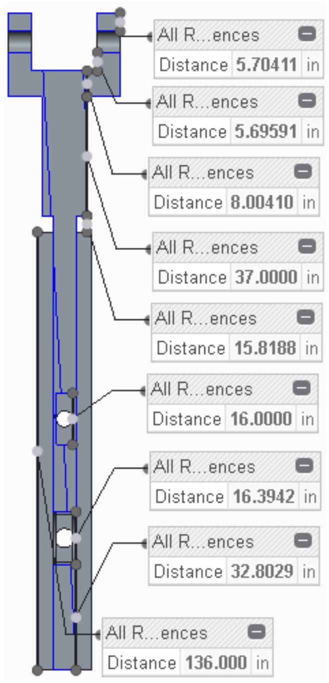
(A)



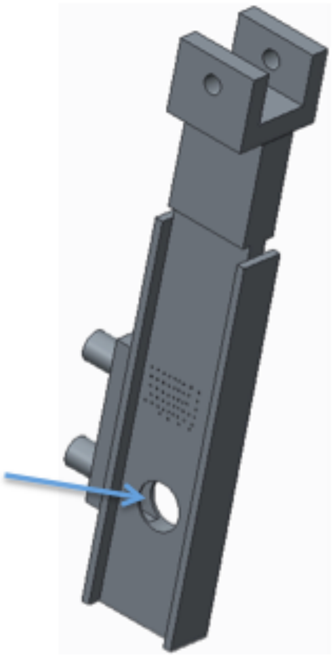
Surf:...E_18)
Diameter 6.10000 in
Radius 3.05000 in
All R...ences
Distance 9.30000 in
Surf:...E_15)
Diameter 5.02000 in
Radius 2.51000 in
All R...ences
Distance 5.00000 in



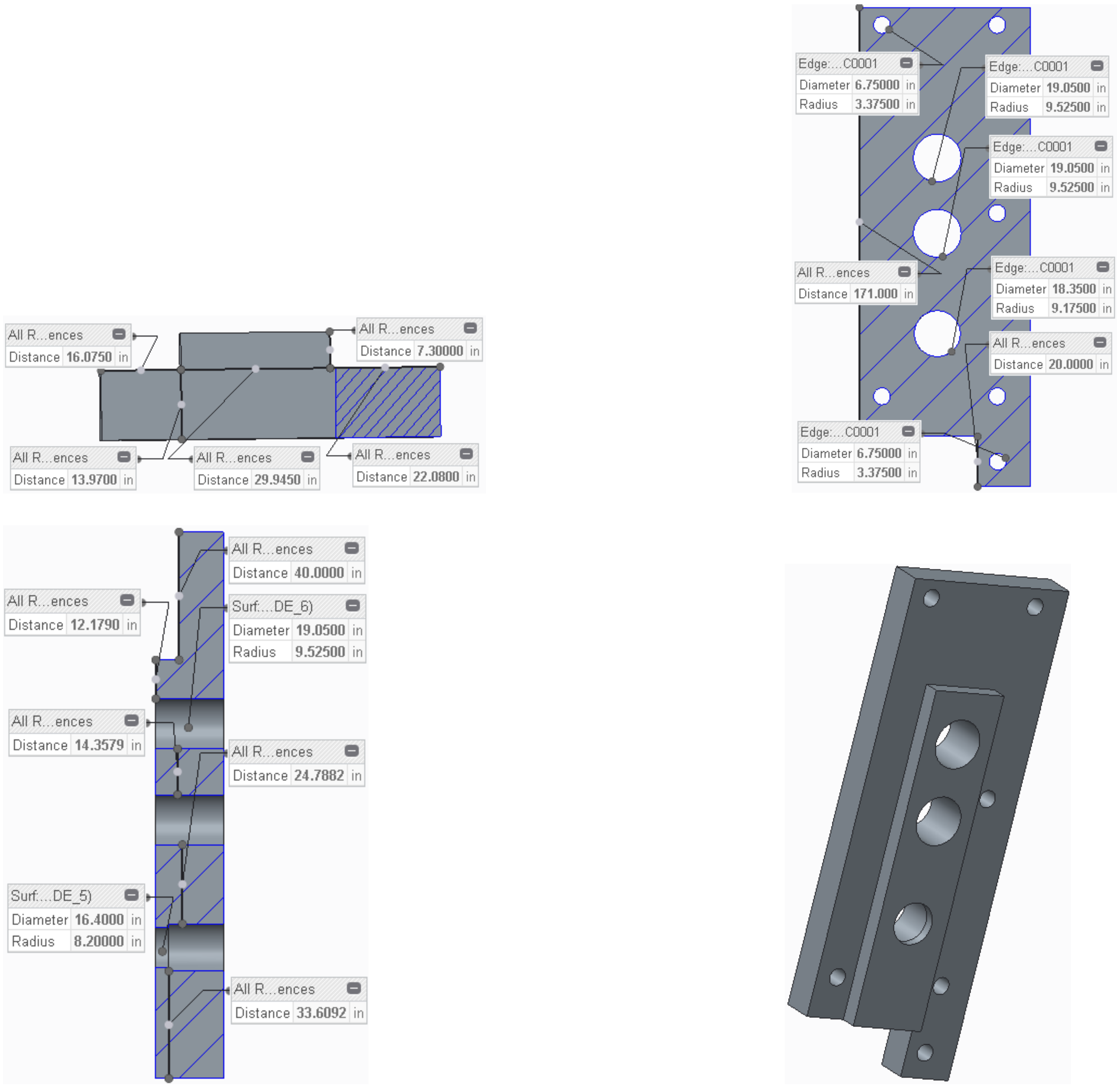
Edge:...E_30)
Diameter 6.60000 in
Radius 3.30000 in
All R...ences
Distance 47.5000 in
All R...ences
Distance 7.64264 in
All R...ences
Distance 57.5000 in
All R...ences
Distance 13.0081 in
All R...ences
Distance 7.66926 in
Edge:...DE_5)
Diameter 16.4000 in
Radius 8.20000 in
All R...ences
Distance 7.64264 in
All R...ences
Distance 2.85000 in
All R...ences
Distance 30.4500 in



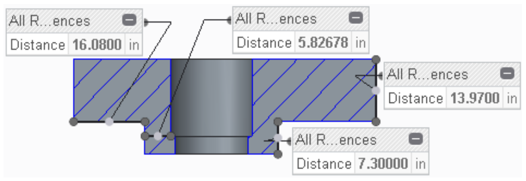
All R...ences
Distance 5.70411 in
All R...ences
Distance 5.69591 in
All R...ences
Distance 8.00410 in
All R...ences
Distance 37.0000 in
All R...ences
Distance 15.8188 in
All R...ences
Distance 16.0000 in
All R...ences
Distance 16.3942 in
All R...ences
Distance 32.8029 in
All R...ences
Distance 136.000 in

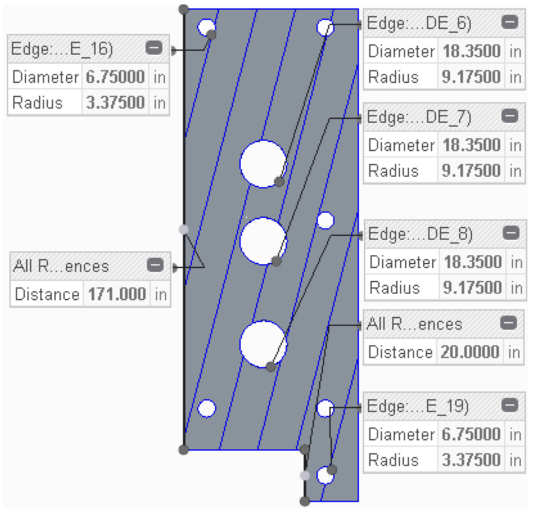


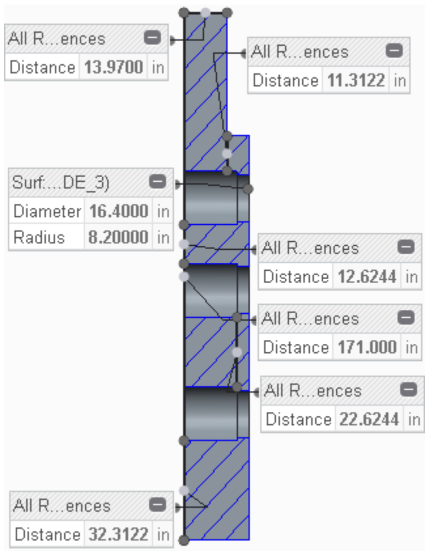
(B)


All R...ences
Distance 16.0750 in
All R...ences
Distance 7.30000 in
All R...ences
Distance 13.9700 in
All R...ences
Distance 29.9450 in
All R...ences
Distance 22.0800 in
Edge:...C0001
Diameter 6.75000 in
Radius 3.37500 in
Edge:...C0001
Diameter 19.0500 in
Radius 9.52500 in
Edge:...C0001
Diameter 19.0500 in
Radius 9.52500 in
All R...ences
Distance 171.000 in
Edge:...C0001
Diameter 18.3500 in
Radius 9.17500 in
All R...ences
Distance 20.0000 in
Edge:...C0001
Diameter 6.75000 in
Radius 3.37500 in
All R...ences
Distance 40.0000 in
All R...ences
Distance 12.1790 in
Surf:...DE_6)
Diameter 19.0500 in
Radius 9.52500 in
All R...ences
Distance 14.3579 in
All R...ences
Distance 24.7882 in
Surf:...DE_5)
Diameter 16.4000 in
Radius 8.20000 in
All R...ences
Distance 33.6092 in

(C)







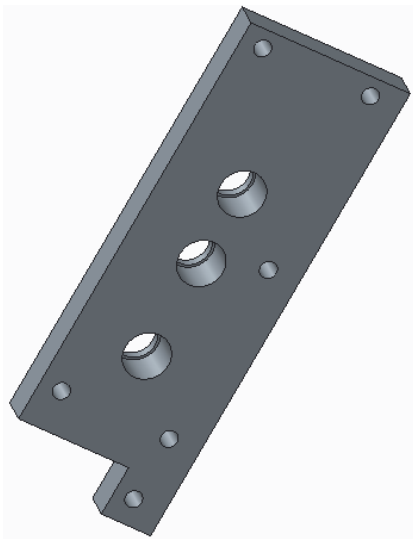
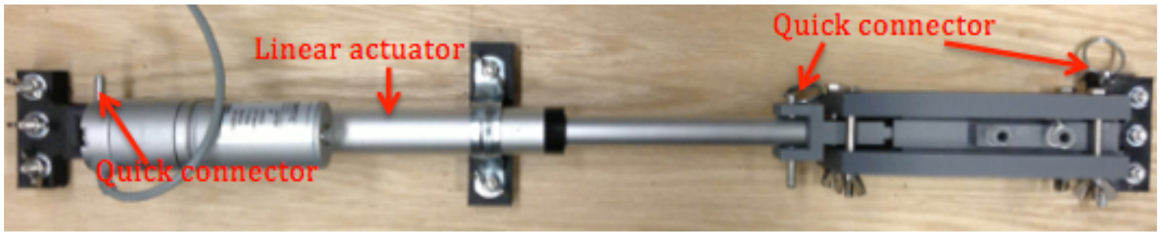


Figure 35. Bottom, front, and left side cross section view of LATTS three 3D parts. We selected interpret dimensions options in PTC creo, which means 1’’ becomes 1mm in the design. (A) LATTS elevator part’s bottom, front, and left side cross section view according from left to right and from top to bottom. (B) LATTS left part of the base’s bottom, front, and left side cross section view from left to right and from top to bottom. (C) LATTS right part of the base’s bottom, front, and left side view from left to right and from top to bottom.

(A)



(B)

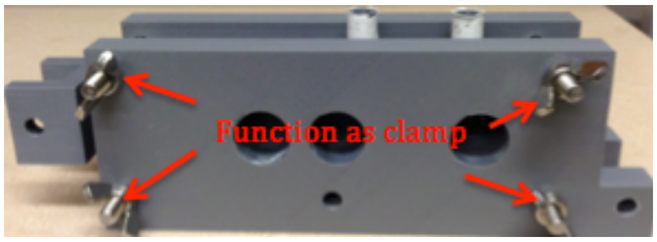


Figure 36. Assembled Real 3D Parts Object of LATTS. (A) A prototype photo of the LATTS for drosophilae training and testing. The main mechanical components of LATTS are composed of the 3D parts: a linear actuator and three quick connectors. The system is mounted on a wooden board. (B) Magnified illustrations showing the 3D parts and we call them together as base.

As shown in the Figure 36A, the linear actuator moves the base's center (elevator) up and down to achieve the experimental requirements. The three quick connectors make moving the apparatus very easy. As shown in Figure 36B, the four bolts with wing nuts create an airtight seal without using a clamp. The apparatus doesn't need to be adjusted for every experiment.

<u>Tube Fittings and Tubing for Both RATTS and LATTS</u>

According to the RATTS and LATTS schematic diagram shown in Figure 3 and as described above, all components of the RATTS and LATTS are connected by tubing carrying compressed air except the system's 3D parts. Tubing in the system corresponds to four standard sizes; 1) outer diameter 3/4 inch, inner diameter 5/8 inch, 2) outer diameter 3/8 inch, inner diameter 1/4 inch, 3) outer diameter 1/4 inch, inner diameter 0.170 inch, and 4) outer diameter 1/8 inch, inner diameter 1/16 inch. The tubing is high-strength clear PVC tubing. Figure 37 shows samples of the four standard sizing tubing used in this project. Most of the fittings are nylon with a mix of barbed and push-to-connect fittings. Figure 38 shows samples of all the fittings used in the system. Figure 39 shows samples of the training tube and the testing tube we used in the system.

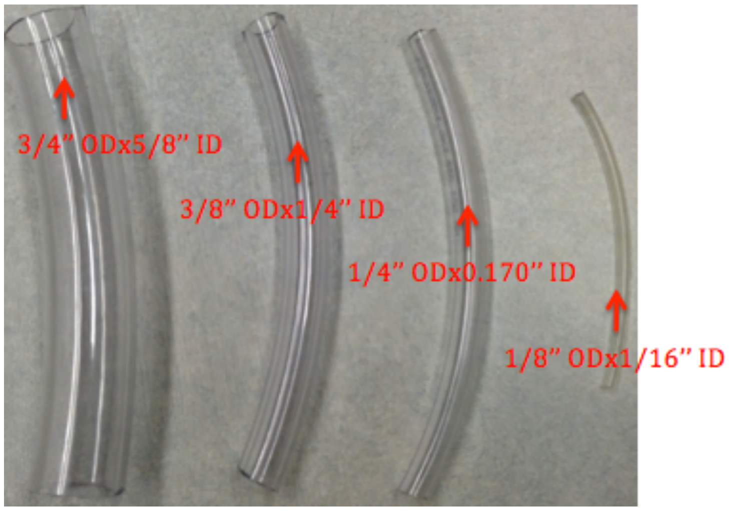


Figure 37. The Four Standard Sizing Tubes

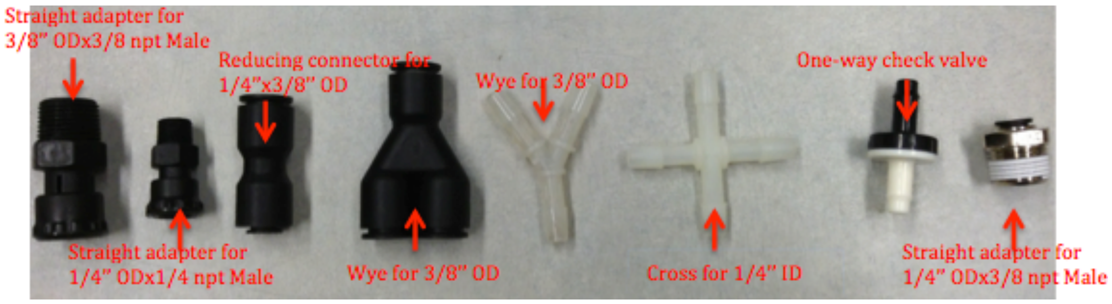

Straight adapter for 3/8" ODx3/8 npt Male
Reducing connector for 1/4"x3/8" OD
Wye for 3/8" OD
One-way check valve
Straight adapter for 1/4" ODx1/4 npt Male
Wye for 3/8" OD
Cross for 1/4" ID
Straight adapter for 1/4" ODx3/8 npt Male

Figure 38. Samples of All the Fittings Used In the System

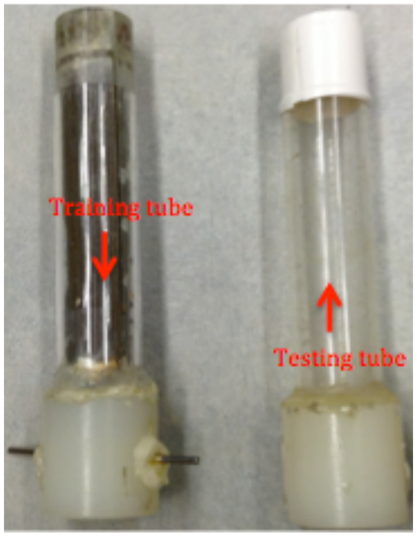


Figure 39. Sample of The Training Tube and The Testing Tube

Chapter IV

Software Development

One of the Arduino Uno microcontrollers was used as a control unit for the RATTS and LATTS, and the other was used as a data log unit for the RATTS and LATTS. Arduino is an open-source electronics prototyping board with a boot loader for program uploading. It consists of both a physical programmable circuit board (often referred to as the microcontroller) and software, or integrated development environment (IDE) that runs on a computer used to write and upload computer code to the physical board. In our project, we used the IDE Arduino and Labview to control the Arduino Uno control unit board. We used IDE Arduino method to control the Arduino Uno data log unit board. A detailed description of the application code developed and the selected programming language is given in the following subsections.

IDE Arduino

The IDE Arduino is a programming environment used to program modules. The IDE uses a modified C language compiler to build, translate and transmit the code to the microcontroller board. This platform offers the ability to make any project that involves different inputs. Through an application developed in Arduino, it can be establish as a specific output. The IDE also has a set of additional code functions called libraries. Libraries extend the commands available to provide capabilities not available in the core Arduino language [7]. In essence, a library allows the user to perform seemingly complex functions using a small set of commands. The SERVO, SD and RTC library are called within the sketch to simplify the code in our project.

The goal of this project is to successfully achieve automated *Drosophila* learning and memory experiments, and have fail-safe tests to ensure that air flow rates and electrical shocks are delivered correctly. The system has two main states known as the “training phase” and “testing phase” for both learning and memory experiments. The two states are shown in Figure 40.

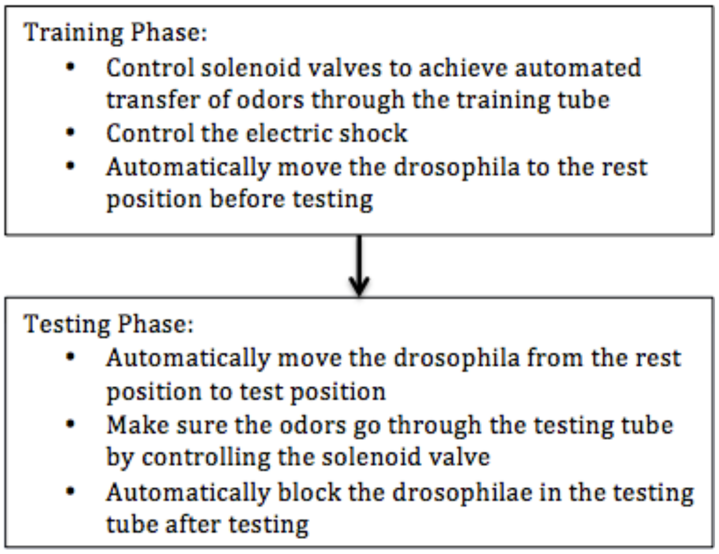

Training Phase:
• Control solenoid valves to achieve automated transfer of odors through the training tube
• Control the electric shock
• Automatically move the drosophila to the rest position before testing
Testing Phase:
• Automatically move the drosophila from the rest position to test position
• Make sure the odors go through the testing tube by controlling the solenoid valve
• Automatically block the drosophilae in the testing tube after testing

Figure 40. Two States of Drosophila Learning and Memory Experiment

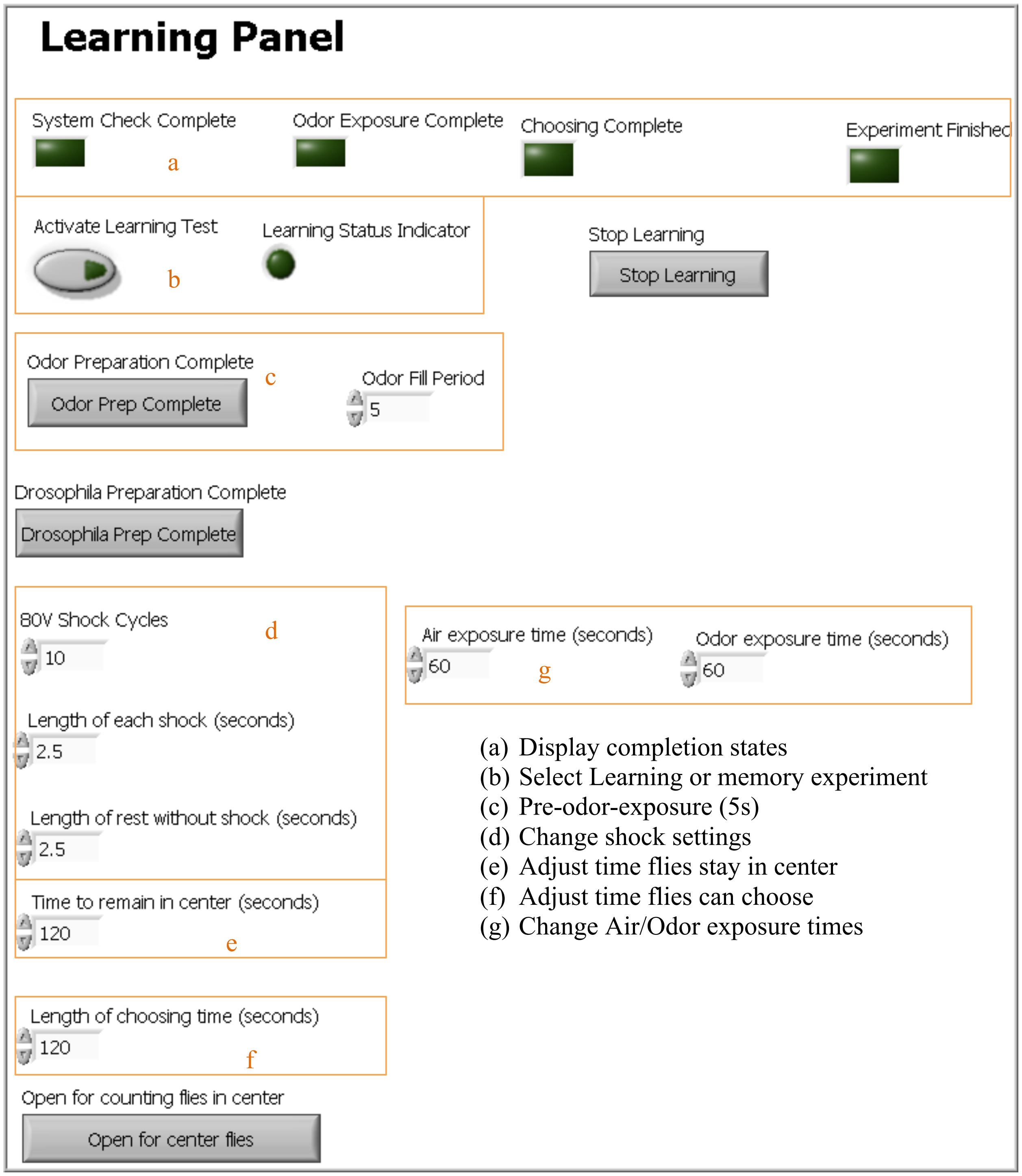
While the system is in these two phases, the data are continuously recorded. The complete sketch source code for both RATTS and LATTS Arduino Uno control unit and Arduino Uno data logger units are provided in appendix B.

LabVIEW

The need to control Arduino requires the use of software designed for this purpose. We can use the IDE Arduino to write the sketch source code and upload to Arduino. We can also implement this using LabVIEW, which allows us to control and automate different parts of the project. Since LabVIEW is based on graphical tools, we can design a system user interface as well, making the system easy for anyone to use.

LabVIEW is an environment based on blocks. Two different parts compose all of the applications developed in this project with LabVIEW: the Front Panel and the Block Diagram. We only use LabVIEW to control the Arduino Uno control unit. Figure 41 shows the Front Panel of our application. The block diagram of RATTS is given in the appendix C.

(A)



80V Shock Cycles: 10

Length of each shock (seconds): 2.5

Length of rest without shock (seconds): 2.5

Time to remain in center (seconds): 120

Length of choosing time (seconds): 120

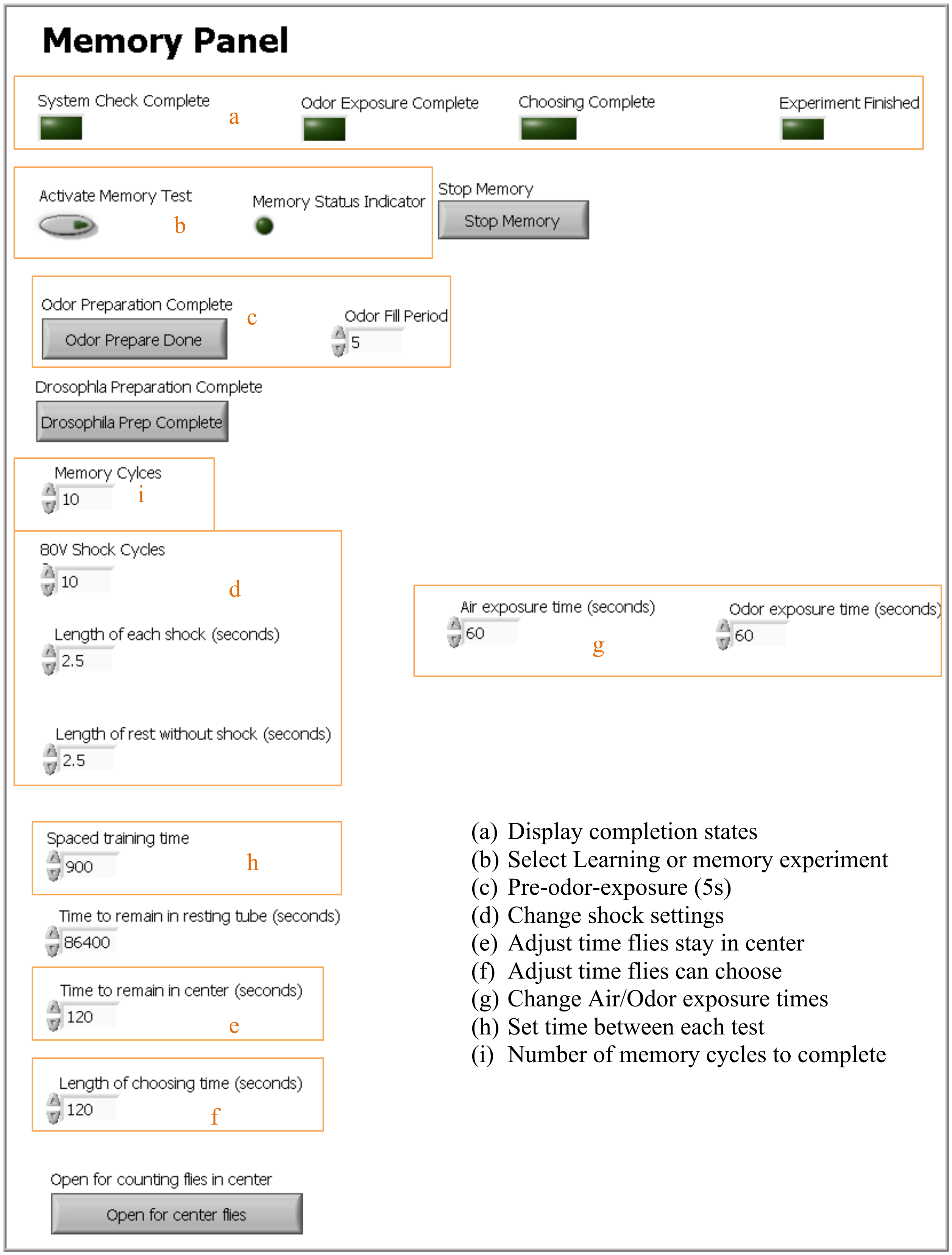
Odor Fill Period: 5

Air exposure time (seconds): 60

Odor exposure time (seconds): 60

(a) Display completion states
(b) Select Learning or memory experiment
(c) Pre-odor-exposure (5s)
(d) Change shock settings
(e) Adjust time flies stay in center
(f) Adjust time flies can choose
(g) Change Air/Odor exposure times

(B)



Memory Panel

System Check Complete

Odor Exposure Complete

Choosing Complete

Experiment Finished

Activate Memory Test

Memory Status Indicator

Stop Memory

Stop Memory

Odor Preparation Complete

Odor Prepare Done

Odor Fill Period: 5

Drosophla Preparation Complete

Drosophila Prep Complete

Memory Cylces: 10

80V Shock Cycles: 10

Length of each shock (seconds): 2.5

Length of rest without shock (seconds): 2.5

Air exposure time (seconds): 60

Odor exposure time (seconds): 60

Spaced training time: 900

Time to remain in resting tube (seconds): 86400

Time to remain in center (seconds): 120

Length of choosing time (seconds): 120

Open for counting flies in center

Open for center flies

(a) Display completion states
(b) Select Learning or memory experiment
(c) Pre-odor-exposure (5s)
(d) Change shock settings
(e) Adjust time flies stay in center
(f) Adjust time flies can choose
(g) Change Air/Odor exposure times
(h) Set time between each test
(i) Number of memory cycles to complete

Figure 41. Front Panel of our application. (A) *Drosophila* learning part experiment user interface. (B) *Drosophila* training part experiment user interface.

Chapter V

Experiment Results

The experiment results obtained with the LATTS are presented. The experiment results for the RATTS are nearly the same as for LATTS because the setup is the same, except for the design of the 3D parts. Therefore, we didn't include RATTS results. This chapter includes the odor sensor data logging results, flow sensor data logging results and electric shock data logging results. All these results clearly show that (1) the system can achieve automatic transition of odors, (2) the system can achieve automated transition experimental phase from training phase to testing phase, and (3) the air flow rate and the electric shock can be delivered correctly.

Experimental Setup

The odorants 4-methylcyclohexanol (MCH) and 3-octanol (OCT) were diluted in mineral oil. Three micro molar of the odorant and three milliliters of mineral oil were loaded into the odor cup. An 80V electric shock was used. The airflow speed through each odor cup was adjusted to 1561mL/min with a flow meter [26] (see Figure 42). The experiment conditions were 24 degree Celsius and 2% humidity (way too low).

Figure 42. Bel-Art 404060020 Riteflow Panel-Mounted 65mm Flowmeter

Results

<u>TSG 2620 Odor Sensor Results</u>

Figure 43 shows the odor concentration in the training tube recorded with the odor sensor (as shown in Figure 3). The square black dots show the odor concentration results with odor delivery sequence of MCH, air and OCT. The red circle dots show the odor concentration results with odor delivery sequence of OCT, air and MCH. Each odor delivery was for one minute to the training tube. According to the data, we can easily find out that the system can achieve automatic transfer of odors, and the odor sensor can provide us a fail-safe test.

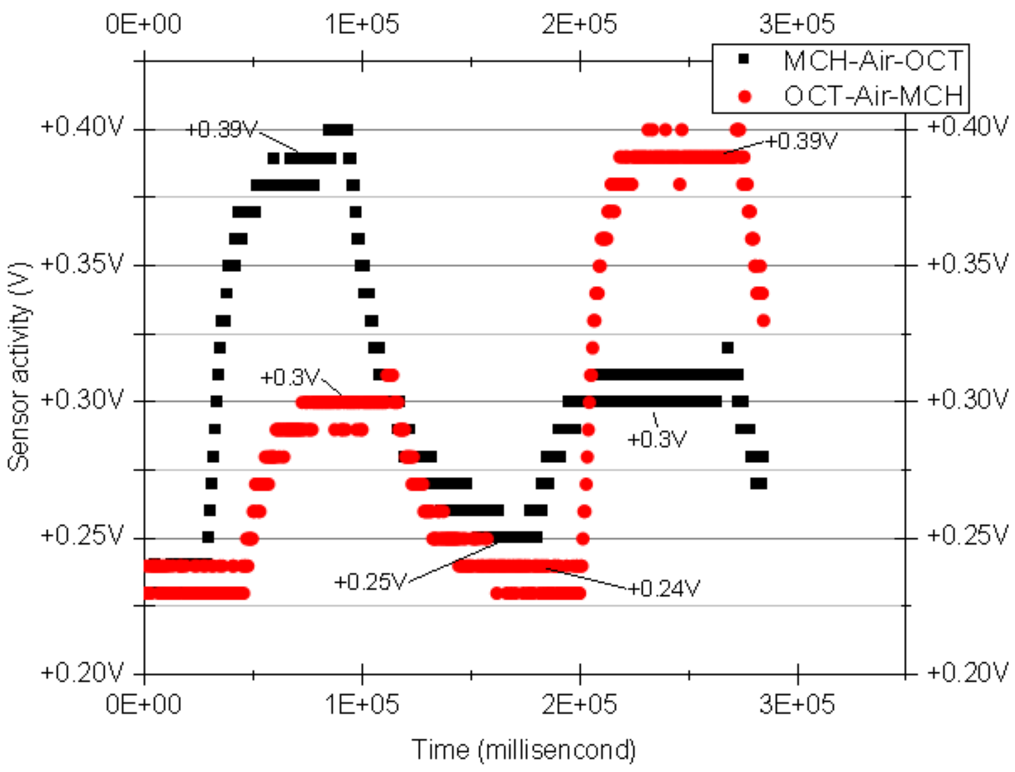


Figure 43. Measurement of actual odor concentrations in the training tube. When the odor sensor detects MCH, the Arduino odor sensor analog input will reach 0.39V and stabilize for a while. When the odor sensor detects Air, the Arduino odor sensor analog input will reach 0.24V-0.25V and stabilize for a while. When the odor sensor detects OCT, the Arduino odor sensor analog input will reach 0.3V and stabilize for a while.

FS5 Flow Sensor Results

There are three thermal mass flow sensor positions shown in Figure 3. The experiment test results are from the flow sensor located near one side of the testing tube, which is on the same side as the training tube (as shown in Figure 44). We did the tests changing the flow rate from 1561mL/min to 1465mL/min, and then from 1465mL/min to 1657mL/min in the testing tube. From Figure 44, one can see that when the flow rate is changed, there is a curve change. The data logging of FS5 flow sensor can prove whether or not the airflow rate is delivered correctly.

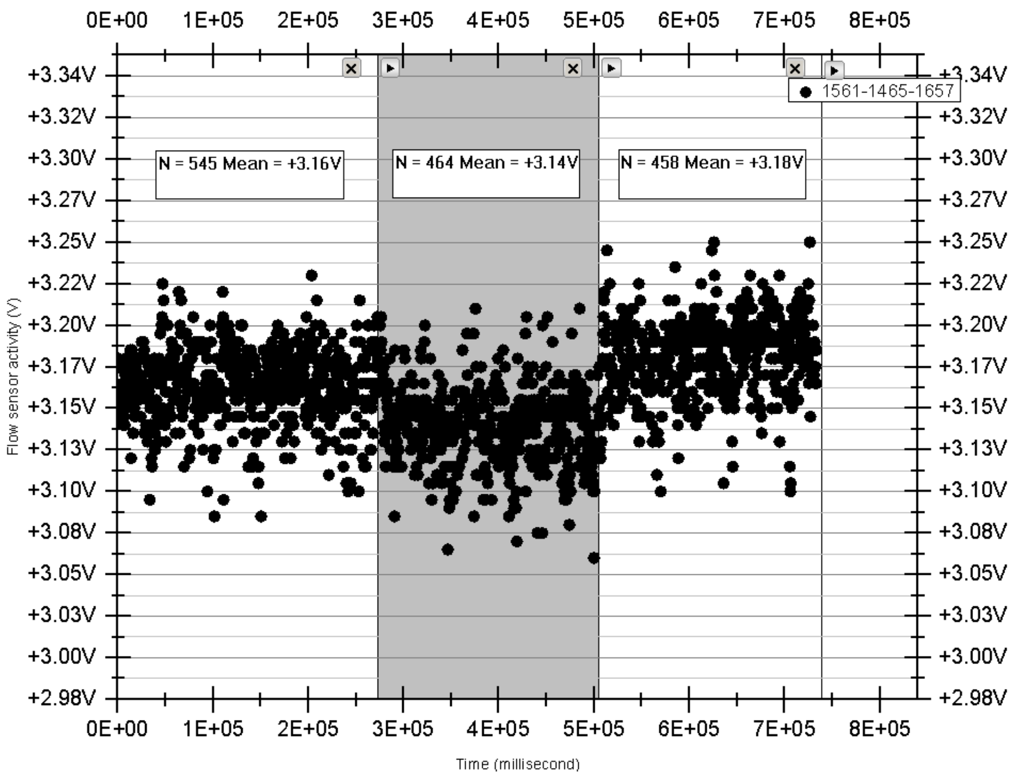
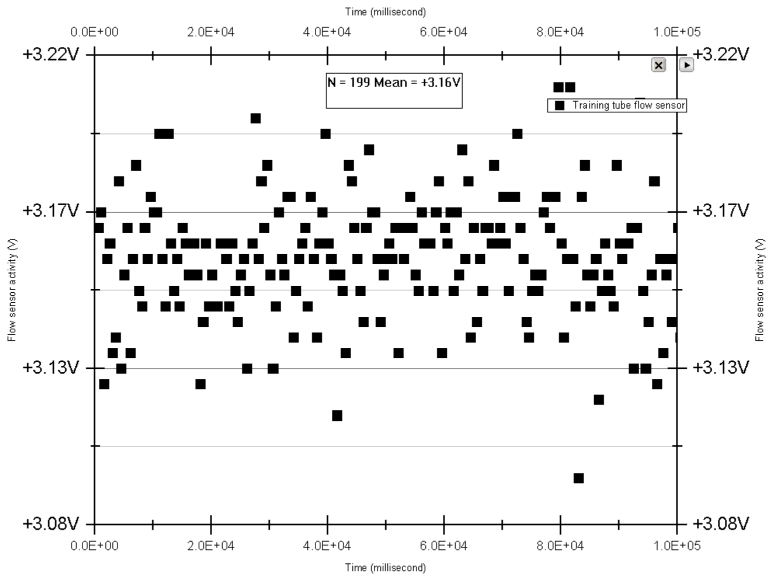


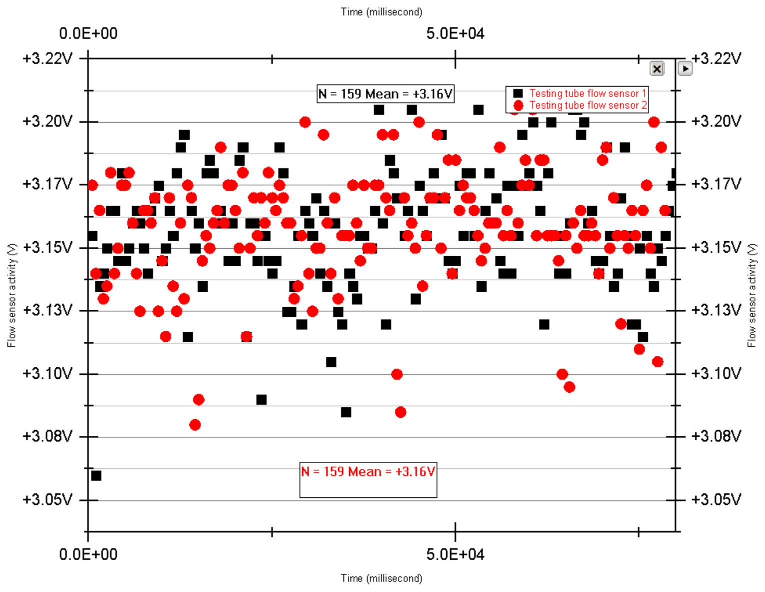
Figure 44. Measurement of Actual Flow Rate of One of The Testing Tubes. When the flow rate is 1561mL/min, the Arduino flow sensor's analog input average voltage is 3.16V. When the flow rate is 1465mL/min, the Arduino flow sensor's analog input average voltage is 3.14V. When the flow rate is 1657mL/min, the Arduino flow sensor's analog input average is 3.18V.

Figure 45 shows when manually adjusted the airflow of training tube and two testing tubes to 1561mL/min and constant, the results of data logging for the three flow sensors. The lack of curve change in Figure 45C shows the air flow rate is being delivered correctly.

(A)



(B)



(C)

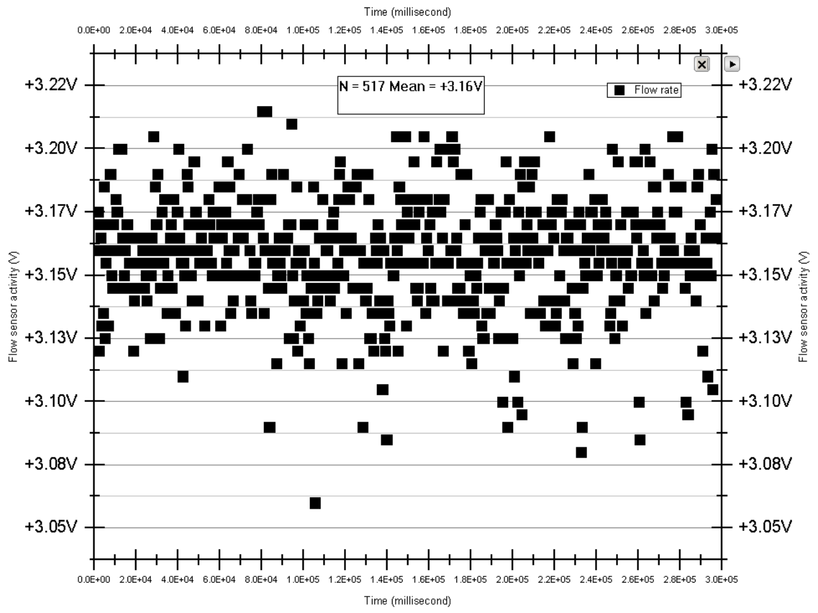


Figure 45. Measurement of the Actual Flow Rate of A Training Tube and Two Testing Tubes. (A) The actual flow rate of the training tube when the speed of airflow through the training tube was adjusted to 1561mL/min with a flow meter [26]. The Arduino flow sensor analog input average voltage is 3.16V. (B) The actual flow rate of the two testing tube when the speed of airflow through both these two testing tubes was adjusted to 1561mL/min with a flow meter. The Arduino flow sensor analog input average voltage for the two testing tubes is both 3.16V. (C) Plotting both (A) and (B) data together in a figure to see if there is a curve change is shown in Figure 44.

Electric Shock Data Logging Results

In our experiments, we electrically shocked *Drosophila* 10 times in the training tube. Figure 46 shows the data log for the electric shocks, recording ten peaks. The electric shock data log proves our electrical shocks were delivered correctly.

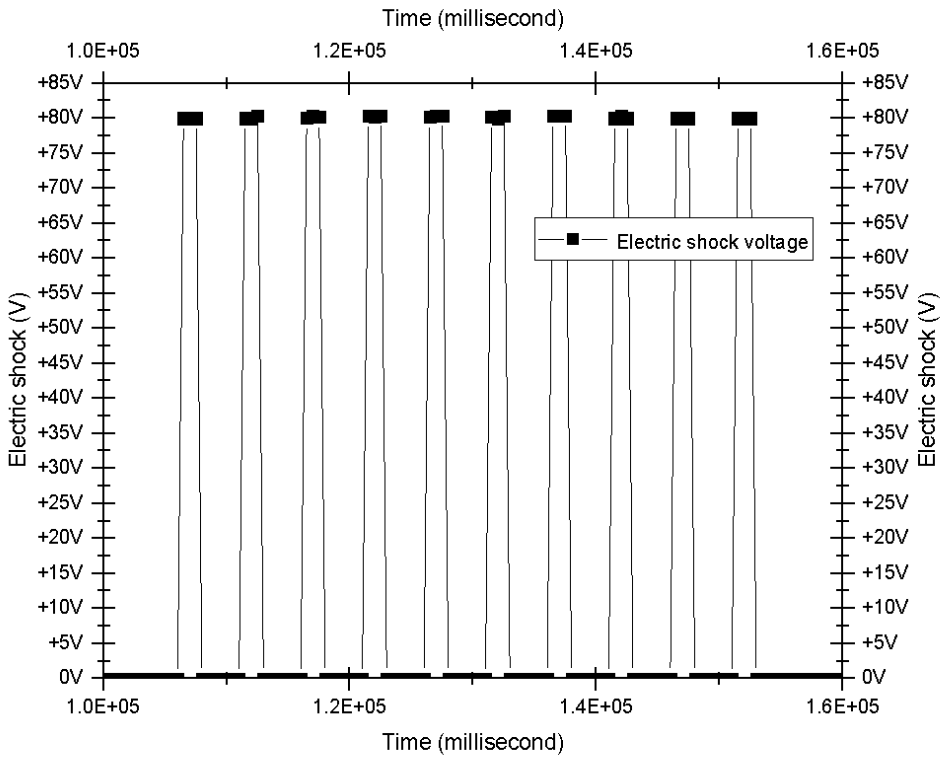


Figure 46. The Electric Shock Data Logging Results

Chapter VI

Conclusion and Further Development

Conclusion

This thesis demonstrates the feasibility of two platform designs using inexpensive hardware and software, for a total design price under $700US per system. The two platforms were designed using inexpensive electronic devices, 3D printed parts and different sizes of the fittings. The two platforms are controlled by an open source Arduino microcontroller, eight solenoid valves, two servos, a linear actuator and a square pulse stimulator. We use IDE Arduino and Labview to control the Arduino Uno control unit board. The two platforms are data logged by an open source Arduino microcontroller, an odor sensor and three flow sensors. We use the IDE Arduino method to data log with the Arduino Uno data logger unit board.

It is important to note that the automated systems developed in this project offer advantages compared with previously designed systems [4], because it does not require any manual operation once the system finishes the preparation stage of the experiment. The two platforms selected allow studying a lot of possibilities related to the communication, circuit design and system control through different tools such as IDE Arduino and LabVIEW. In conclusion, the development of this project has allowed expansion of interdisciplinary knowledge in different areas of the technology.

Future Development

There are plans to machine the final parts from acrylic material using a laser cutter, based on the 3D printed prototype to obtain multiple systems for testing. The Broadie Lab is interested in employing multiple systems at once, so incorporating stacked systems is in the works for imminent completion. The possibility of powering more than one system with each linear actuator has also been explored, and is still being considered for final implementation. The functionality of this idea may depend on the final placement of the systems. The training/testing apparatus must be contained because of the humidity requirements of the experiment, so a box has been designed to isolate the electrical components from the humidity. Development of a feedback system is also planned to allow for control of the flow rate and odor concentration. Finally, a notification system may be implemented to notify testers when the experiment is done.

APPENDIX A

Schematic Diagram

The Figure 47 is the control circuit schematic diagram for all of the electronics controlled in RATTS. The electronics for controlling RATTS include eight solenoid valves, two servomotors, and a square pulse stimulator. The Figure 48 is the control circuit schematic diagram for all of the electronics controlled in LATTS. The electronics for controlling LATTS include eight solenoid valves, a linear actuator, and a square pulse stimulator. The Figure 49 is the data logging circuit schematic diagram for all of the electronics data logging in RATTS and LATTS. The electronics for data logging include an odor sensor and three flow sensors.

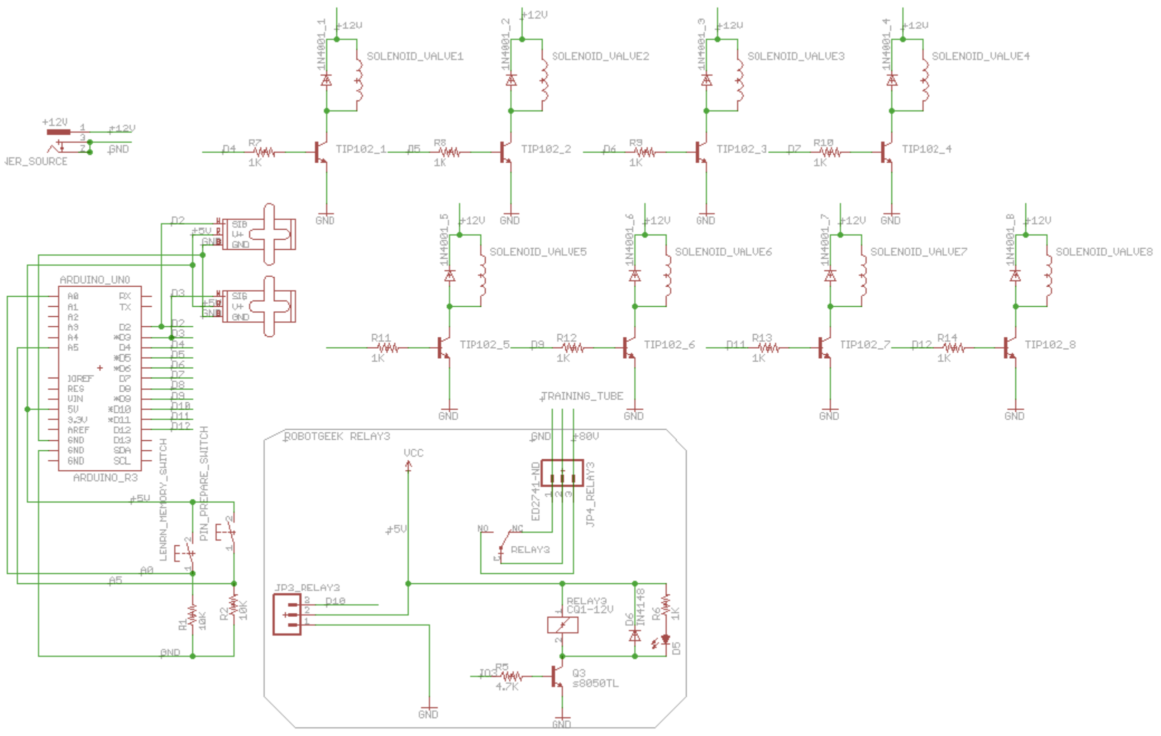


Figure 47. RATTS Control Circuit Schematic Diagram

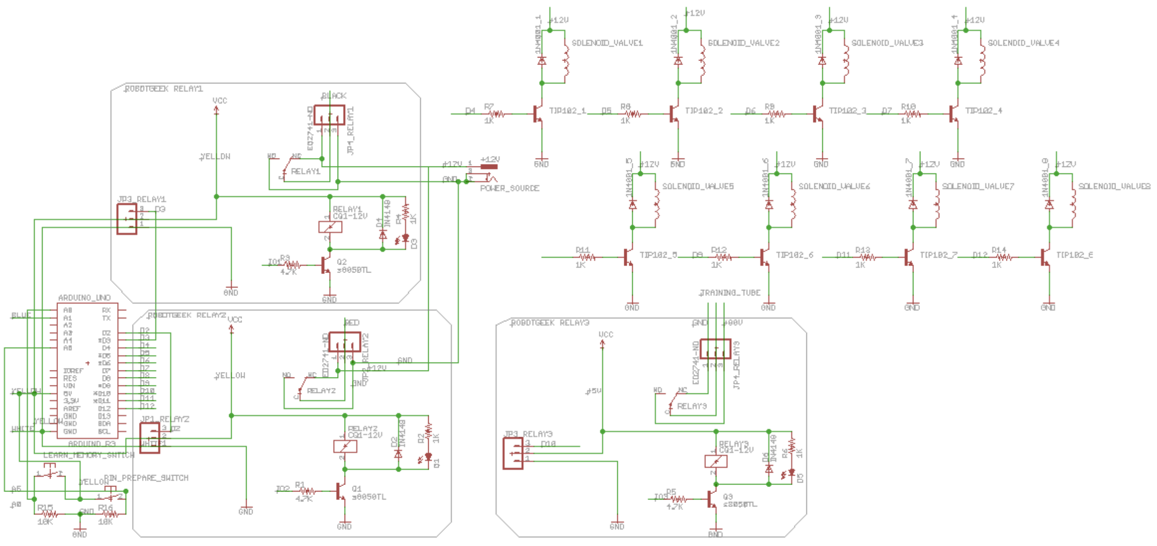


Figure 48. LATTS Control Circuit Schematic Diagram

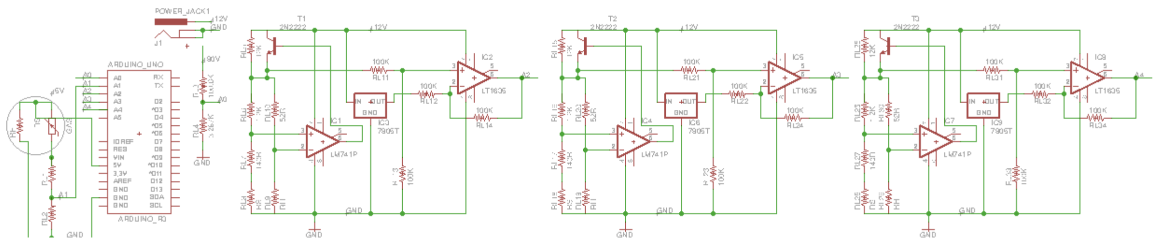


Figure 49. RATTS and LATTS Data logging Circuit Schematic Diagram

APPENDIX B

Arduino Uno Control Unit Code Notes of Both RATTS and LATTS

Arduino Uno Control Unit Code Notes of RATTS

The application code uploaded to the Arduino Uno control unit is detailed below:

```
#include <Servo.h>  //Servo Class
//Initial time delay period between servo motors and solenoid valves
const int initalTimeDelay = 1000;
const int moveTimeDelay = 15000;
const int stayCenterTimeDelay = 120;
const int chooseTimeDelay = 120;
const int rotateNeedTimeDelay = 1500;
const int prepareTimeDelay = 1000;
const int drosophilaFlyTimeDelay = 3500;
const int odorFillPeriod = 5000;

//Drosophila learning and memory choice
const int buttonPinLearnMemoryChoice = 14;
int buttonStateLearnMemoryChoice = 0;

//Prepare done pushbutton
const int buttonPinPrepare = 19;
int buttonStatePrepare = 0;

//Up servo and down servo parameters
int servoPinUp = 3;
int servoPinDown = 2;

Servo servoUp;
Servo servoDown;// create servo object to control a servo

int angle = 0;

//Initialize all the input and output pins
void setup()
{
  pinMode(buttonPinLearnMemoryChoice, INPUT);
  pinMode(buttonPinPrepare, INPUT);
  servoUp.attach(servoPinUp);
  servoDown.attach(servoPinDown);  // attaches the servo on pin 9 to the servo object
  pinMode(13, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(4, OUTPUT);
}

void loop() {
  //When the learning and memory choose pushbutton didn't push, the drosophila is doing
  //memory experimental. When the learning and memory choose pushbutton pushed,
  //the drosophila is doing learning experimental.
  buttonStateLearnMemoryChoice = digitalRead(buttonPinLearnMemoryChoice);
  while(buttonPinLearnMemoryChoice == LOW) {
    initialServoSolenoid();
    odorPrepareDone();
    odorFill();
    drosophilaPrepareDone();
    for(int memoryCycle=1; memoryCycle<=10; memoryCycle++) {
        oneCycleTraining();
        fifteenAirBlank();
      }
    moveTrainingToResting();
    restingPeriod();
    moveRestingToCenter();
    stayCenterPeriod();
    choosePeriod();
    stopChoosePeriod();
    centerToRestingForLearning();
    }

    initialServoSolenoid();
    odorPrepareDone();
    odorFill();
    drosophilaPrepareDone();
    oneCycleTraining();
    moveTrainingToCenter();
    stayCenterPeriodForLearning();
    choosePeriod();
    stopChoosePeriod();
    centerToRestingForLearning();
}

//First start, the system go through the servo motors and solenoid valves first setting
void initialServoSolenoid() {
  //Check all the Solenoid Valve is working
  digitalWrite(13, HIGH);
  delay(initalTimeDelay);
  digitalWrite(12, HIGH);    // turn the Powertail off
```

```
delay(initalTimeDelay);
digitalWrite(11, HIGH); // Turn the Powertail on
delay(initalTimeDelay);
digitalWrite(9, HIGH);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(8, HIGH); // Turn the Powertail on
delay(initalTimeDelay);
digitalWrite(7, HIGH);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(6, HIGH); // Turn the Powertail on
delay(initalTimeDelay);
digitalWrite(5, HIGH);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(4, HIGH);    // turn the Powertail off
delay(initalTimeDelay);


 //Solenoid valve initial position
digitalWrite(13, LOW);
delay(initalTimeDelay);
digitalWrite(12, LOW);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(11, LOW); // Turn the Powertail on
delay(initalTimeDelay);
digitalWrite(9, LOW);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(8, LOW); // Turn the Powertail on
delay(initalTimeDelay);
digitalWrite(7, LOW);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(6, LOW); // Turn the Powertail on
delay(initalTimeDelay);
digitalWrite(5, LOW);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(4, LOW);    // turn the Powertail off
delay(initalTimeDelay);

//Servo initial position
//1
for(angle = 50; angle<=162; angle++)
{
  servoUp.write(angle);
  delay(1);
}
delay(initalTimeDelay);

//2
for(angle = 162; angle>=57; angle--)
{
  servoUp.write(angle);
  delay(1);
}
delay(initalTimeDelay);

//3
for(angle = 57; angle<=124; angle++)
{
 servoUp.write(angle);
 delay(1);
}
delay(initalTimeDelay);

//4
for(angle = 124; angle<=163; angle++)
{
 servoUp.write(angle);
 delay(1);
}
delay(initalTimeDelay);

//5
for(angle = 163;angle>=104; angle--)
{
 servoUp.write(angle);
 delay(1);
}
delay(initalTimeDelay);

//6
for(angle = 104; angle<=149; angle++)
{
 servoUp.write(angle);
 delay(1);
}

for(angle = 77; angle>=59; angle--)
{
 servoDown.write(angle);
 delay(1);
}
delay(initalTimeDelay);

//8
for(angle = 59; angle<=136; angle++)
{
 servoDown.write(angle);
 delay(1);
}
delay(initalTimeDelay);

//9
for(angle = 149; angle>=60; angle--)
```

```
  {
   servoUp.write(angle);
   delay(1);
  }
  delay(initalTimeDelay);

   //3
  for(angle = 60; angle<=125; angle++)
  {
   servoUp.write(angle);
   delay(1);
  }
  delay(initalTimeDelay);

  digitalWrite(13, HIGH);
}

//Odor prepare done
void odorPrepareDone() {
  buttonStatePrepare = digitalRead(buttonPinPrepare);
  while(buttonStatePrepare == LOW) {
    delay(prepareTimeDelay);
    buttonStatePrepare = digitalRead(buttonPinPrepare);
  }
}

//Since distance from odor cup to training tube
//compare to maunual system is too long, fill some distance odor
void odorFill(){
  digitalWrite(13, LOW);
  digitalWrite(12, HIGH);    // turn the Powertail off
  delay(odorFillPeriod);

  digitalWrite(12, LOW);
  digitalWrite(9, HIGH);
  delay(odorFillPeriod);

  digitalWrite(9, LOW);
  digitalWrite(11, HIGH);
  delay(odorFillPeriod);
  delay(odorFillPeriod);

  digitalWrite(13, HIGH);
}

void drosophilaPrepareDone() {
   buttonStatePrepare = digitalRead(buttonPinPrepare);
  while(buttonStatePrepare == LOW) {
    delay(prepareTimeDelay);
    buttonStatePrepare = digitalRead(buttonPinPrepare);
  }
```

```
}


//One cycle for training the drosophila
void oneCycleTraining() {
  //1
  digitalWrite(13, LOW);
  digitalWrite(11, LOW);
  digitalWrite(12, HIGH);
  for(int shockCycle=1; shockCycle<=10; shockCycle++) {
    eightyShockControl();
  }

  //2
  digitalWrite(12, LOW);
  digitalWrite(11, HIGH);
  for(int i=0; i<60; i++) {
    delay(1000);
  }

  //3
  digitalWrite(11, LOW);
  digitalWrite(9, HIGH);
  for(int i=0; i<60; i++) {
    delay(1000);
  }
}

//Pin 13 for control the 80v shock
void eightyShockControl() {
  digitalWrite(10, HIGH);
  delay(2500);
  digitalWrite(10, LOW);
  delay(2500);
}

//15 minute air blank in between each cycle
void fifteenAirBlank() {
  digitalWrite(9, LOW);
  digitalWrite(11, HIGH);
  for(int i=1; i<=900; i++) {
    delay(1000);
  }
}

//Moving drosophila from training tube to resting tube
void moveTrainingToResting() {
  digitalWrite(11, LOW);
  for(angle = 125; angle>=57; angle--)
  {
    servoUp.write(angle);
    delay(1);
  }
  delay(rotateNeedTimeDelay);
  digitalWrite(8, HIGH);
  digitalWrite(4, HIGH);
  delay(moveTimeDelay);
}

//Move from training tube to the center
void moveTrainingToCenter() {
  digitalWrite(9, LOW);
  for(angle = 125; angle<=163; angle++)
  {
    servoUp.write(angle);
    delay(1);
  }
  delay(rotateNeedTimeDelay);
  digitalWrite(8, HIGH);
  digitalWrite(4, HIGH);
  delay(moveTimeDelay);
}

//Stay center for learning training
void stayCenterPeriodForLearning() {
  digitalWrite(8, LOW);
  digitalWrite(4, LOW);
  digitalWrite(11, HIGH);
 // delay(drosophilaFlyTimeDelay);
 for(angle = 163 ;angle>=104; angle--)
  {
    servoUp.write(angle);
    delay(1);
  }
  delay(rotateNeedTimeDelay);
  for(int i=1;i<= stayCenterTimeDelay ;i++) {
  delay(1000);
  }
}

//Resting in the tube
void restingPeriod() {
  digitalWrite(8, LOW);
  for(angle = 57; angle<=124; angle++)
  {
    servoUp.write(angle);
    delay(1);
  }
  delay(rotateNeedTimeDelay);
  for(int i=1; i<=86400; i++) {
    delay(1000);
  }
```

```
}

//move drosophila from resting tube to the 3d inner part
void moveRestingToCenter() {
  for(angle = 124; angle<=163; angle++)
  {
    servoUp.write(angle);
    delay(1);
  }
  delay(rotateNeedTimeDelay);
  digitalWrite(4, LOW);
  digitalWrite(6, HIGH);
  delay(moveTimeDelay);
}

//Stay in center period
void stayCenterPeriod() {
  digitalWrite(6, LOW);
  for(angle = 163 ;angle>=104; angle--)
  {
    servoUp.write(angle);
    delay(1);
  }
  delay(rotateNeedTimeDelay);
  digitalWrite(11, HIGH);
  delay(stayCenterTimeDelay);
}

//Drosophila choosing time
void choosePeriod() {
  digitalWrite(11, LOW);
  digitalWrite(7, HIGH);
  digitalWrite(5, HIGH);
  for(angle = 104; angle<=149; angle++)
  {
    servoUp.write(angle);
    delay(1);
  }
  for(angle = 77; angle>=59; angle--)
  {
    servoDown.write(angle);
    delay(1);
  }
  for(int i=1;i<=chooseTimeDelay;i++) {
   delay(1000);
  }
}

//Stop choose period
void stopChoosePeriod() {
  for(angle = 59; angle<=136; angle++)
  {
    servoDown.write(angle);
    delay(1);
  }
  delay(rotateNeedTimeDelay);
  digitalWrite(7, LOW);
  digitalWrite(5, LOW);
  digitalWrite(13, HIGH);
  buttonStatePrepare = digitalRead(buttonPinPrepare);
  while(buttonStatePrepare == LOW) {
    delay(prepareTimeDelay);
    buttonStatePrepare = digitalRead(buttonPinPrepare);
  }
}

void centerToRestingForLearning() {
  digitalWrite(13, LOW);
  for(angle = 149; angle>=60; angle--)
  {
    servoUp.write(angle);
    delay(1);
  }
  delay(rotateNeedTimeDelay);
  digitalWrite(8, HIGH);
  digitalWrite(4, HIGH);
  delay(moveTimeDelay);
  digitalWrite(8, LOW);
  digitalWrite(4, LOW);
  for(angle = 60; angle<=125; angle++)
  {
    servoUp.write(angle);
    delay(1);
  }
  delay(rotateNeedTimeDelay);
  digitalWrite(13, HIGH);
  while(1 == 1) {
    delay(300000);
  }
}
```

Arduino Uno Control Unit Code Notes of LATTS

The application code uploaded to the Arduino Uno control unit is detailed below:

```
//Initial time delay period between linear actuator and solenoid valves
const int initalTimeDelay = 1000;
const int moveTimeDelay = 4000;
const int stayCenterTimeDelay = 120;
const int chooseTimeDelay = 120;
const int rotateNeedTimeDelay = 1500;
const int prepareTimeDelay = 1000;
const int drosophilaFlyTimeDelay = 3500;
const int odorFillPeriod = 5000;

//Drosophila learning and memory choice
const int buttonPinLearnMemoryChoice = 14;
int buttonStateLearnMemoryChoice = 0;

//Prepare done pushbutton
const int buttonPinPrepare = 19;
int buttonStatePrepare = 0;

//Up servo and down servo parameters
const int relay1Pin = 3;
const int relay2Pin = 2;
const int sensorPin = 15;

int goalPosition = 350;
int CurrentPosition = 0;

//Initialize all the input and output pins
void setup()
{
  //Initialize the relay pin as an output:
  pinMode(relay1Pin, OUTPUT);
  pinMode(relay2Pin, OUTPUT);

  //Preset the relays to LOW
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);


  pinMode(buttonPinLearnMemoryChoice, INPUT);
  pinMode(buttonPinPrepare, INPUT);
  pinMode(13, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(4, OUTPUT);
}

void loop() {
  //When the learning and memory choose pushbutton didn't push, the drosophila is doing
  //memory experimental. When the learning and memory choose pushbutton pushed,
  //the drosophila is doing learning experimental.
  buttonStateLearnMemoryChoice = digitalRead(buttonPinLearnMemoryChoice);
  while(buttonPinLearnMemoryChoice == LOW) {
    initialServoSolenoidMemory();
    odorPrepareDone();
    odorFill();
    drosophilaPrepareDone();
    for(int memoryCycle=1; memoryCycle<=10; memoryCycle++) {
        oneCycleTraining();
        fifteenAirBlank();
      }
    moveTrainingToResting();
    restingPeriod();
    moveRestingToCenter();
    stayCenterPeriod();
    choosePeriod();
    stopChoosePeriod();
    }

    initialServoSolenoid();
    odorPrepareDone();
    odorFill();
    drosophilaPrepareDone();
    oneCycleTraining();
    moveTrainingToCenter();
    stayCenterPeriodForLearning();
    choosePeriod();
    stopChoosePeriod();
}

void initialServoSolenoid() {
  //Check all the Solenoid Valve is working
  digitalWrite(13, HIGH);
  delay(initalTimeDelay);
  digitalWrite(12, HIGH);    // turn the Powertail off
  delay(initalTimeDelay);
  digitalWrite(11, HIGH); // Turn the Powertail on
  delay(initalTimeDelay);
  digitalWrite(9, HIGH);    // turn the Powertail off
  delay(initalTimeDelay);
  digitalWrite(8, HIGH); // Turn the Powertail on
  delay(initalTimeDelay);
  digitalWrite(7, HIGH);    // turn the Powertail off
  delay(initalTimeDelay);
  digitalWrite(6, HIGH); // Turn the Powertail on
  delay(initalTimeDelay);
  digitalWrite(5, HIGH);    // turn the Powertail off
  delay(initalTimeDelay);
  digitalWrite(4, HIGH);    // turn the Powertail off
  delay(initalTimeDelay);
```

```
 //Solenoid valve initial position
digitalWrite(13, LOW);
delay(initalTimeDelay);
digitalWrite(12, LOW);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(11, LOW); // Turn the Powertail on
delay(initalTimeDelay);
digitalWrite(9, LOW);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(8, LOW); // Turn the Powertail on
delay(initalTimeDelay);
digitalWrite(7, LOW);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(6, LOW); // Turn the Powertail on
delay(initalTimeDelay);
digitalWrite(5, LOW);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(4, LOW);    // turn the Powertail off
delay(initalTimeDelay);

digitalWrite(8, HIGH);
//linear actuator adjust
//1
CurrentPosition = analogRead(sensorPin);
goalPosition = 750;

while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
while(goalPosition < CurrentPosition) {
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, HIGH);
  CurrentPosition = analogRead(sensorPin);
}

  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);
  delay(initalTimeDelay);

 //2
CurrentPosition = analogRead(sensorPin);
goalPosition = 500;

while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
```

```
while(goalPosition < CurrentPosition) {
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, HIGH);
  CurrentPosition = analogRead(sensorPin);
}

  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);
  delay(initalTimeDelay);

//3.1
CurrentPosition = analogRead(sensorPin);
goalPosition = 540;

while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
while(goalPosition < CurrentPosition) {
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, HIGH);
  CurrentPosition = analogRead(sensorPin);
}

  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);
  delay(initalTimeDelay);

//3.2
CurrentPosition = analogRead(sensorPin);
goalPosition = 680;

while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
while(goalPosition < CurrentPosition) {
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, HIGH);
  CurrentPosition = analogRead(sensorPin);
}

  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);
  delay(initalTimeDelay);

//4
CurrentPosition = analogRead(sensorPin);
goalPosition = 810;
```

```
while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
while(goalPosition < CurrentPosition) {
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, HIGH);
  CurrentPosition = analogRead(sensorPin);
}

  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);
  delay(initalTimeDelay);

  digitalWrite(5, HIGH);
  digitalWrite(7, HIGH);
  digitalWrite(13, HIGH);
  odorPrepareDone();
  digitalWrite(13, LOW);

  //5
CurrentPosition = analogRead(sensorPin);
goalPosition = 640;

while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
while(goalPosition < CurrentPosition) {
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, HIGH);
  CurrentPosition = analogRead(sensorPin);
}

  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);
  delay(initalTimeDelay);


//6
CurrentPosition = analogRead(sensorPin);
goalPosition = 780;

while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
```

```
 while(goalPosition < CurrentPosition) {
   digitalWrite(relay1Pin, LOW);
   digitalWrite(relay2Pin, HIGH);
   CurrentPosition = analogRead(sensorPin);
 }

   digitalWrite(relay1Pin, LOW);
   digitalWrite(relay2Pin, LOW);
   delay(initalTimeDelay);

   digitalWrite(5, LOW);
   digitalWrite(7, LOW);
   digitalWrite(12, HIGH);
   digitalWrite(13, HIGH);
   odorPrepareDone();
   digitalWrite(13, LOW);
   digitalWrite(12, LOW);
   digitalWrite(11, HIGH);
   digitalWrite(13, HIGH);
   delay(3000);
   odorPrepareDone();
   digitalWrite(13, LOW);
   digitalWrite(11, LOW);
   digitalWrite(9, HIGH);
   digitalWrite(13, HIGH);
   delay(3000);


}


//First start, the system go through the servo motors and solenoid valves first setting
void initialServoSolenoid() {
 //Check all the Solenoid Valve is working
 digitalWrite(13, HIGH);
 delay(initalTimeDelay);
 digitalWrite(12, HIGH);    // turn the Powertail off
 delay(initalTimeDelay);
 digitalWrite(11, HIGH); // Turn the Powertail on
 delay(initalTimeDelay);
 digitalWrite(9, HIGH);    // turn the Powertail off
 delay(initalTimeDelay);
 digitalWrite(8, HIGH); // Turn the Powertail on
 delay(initalTimeDelay);
 digitalWrite(7, HIGH);    // turn the Powertail off
 delay(initalTimeDelay);
 digitalWrite(6, HIGH); // Turn the Powertail on
 delay(initalTimeDelay);
 digitalWrite(5, HIGH);    // turn the Powertail off
 delay(initalTimeDelay);
 digitalWrite(4, HIGH);    // turn the Powertail off
```

```
delay(initalTimeDelay);


 //Solenoid valve initial position
digitalWrite(13, LOW);
delay(initalTimeDelay);
digitalWrite(12, LOW);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(11, LOW); // Turn the Powertail on
delay(initalTimeDelay);
digitalWrite(9, LOW);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(8, LOW); // Turn the Powertail on
delay(initalTimeDelay);
digitalWrite(7, LOW);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(6, LOW); // Turn the Powertail on
delay(initalTimeDelay);
digitalWrite(5, LOW);    // turn the Powertail off
delay(initalTimeDelay);
digitalWrite(4, LOW);    // turn the Powertail off
delay(initalTimeDelay);

digitalWrite(8, HIGH);
//linear actuator adjust
//1
CurrentPosition = analogRead(sensorPin);
goalPosition = 750;

while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
while(goalPosition < CurrentPosition) {
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, HIGH);
  CurrentPosition = analogRead(sensorPin);
}

  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);
  delay(initalTimeDelay);

 //2
CurrentPosition = analogRead(sensorPin);
goalPosition = 500;

while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
while(goalPosition < CurrentPosition) {
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, HIGH);
  CurrentPosition = analogRead(sensorPin);
}

  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);
  delay(initalTimeDelay);


//3
CurrentPosition = analogRead(sensorPin);
goalPosition = 680;

while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
while(goalPosition < CurrentPosition) {
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, HIGH);
  CurrentPosition = analogRead(sensorPin);
}

  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);
  delay(initalTimeDelay);

//4
CurrentPosition = analogRead(sensorPin);
goalPosition = 810;

while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
while(goalPosition < CurrentPosition) {
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, HIGH);
  CurrentPosition = analogRead(sensorPin);
}

  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);
  delay(initalTimeDelay);
  digitalWrite(5, HIGH);
  digitalWrite(7, HIGH);
  digitalWrite(13, HIGH);
  odorPrepareDone();
  digitalWrite(13, LOW);

  //5
CurrentPosition = analogRead(sensorPin);
goalPosition = 640;

while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
while(goalPosition < CurrentPosition) {
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, HIGH);
  CurrentPosition = analogRead(sensorPin);
}

  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);
  delay(initalTimeDelay);


//6
CurrentPosition = analogRead(sensorPin);
goalPosition = 780;

while(goalPosition > CurrentPosition) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, LOW);
  CurrentPosition = analogRead(sensorPin);
}
while(goalPosition < CurrentPosition) {
  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, HIGH);
  CurrentPosition = analogRead(sensorPin);
}

  digitalWrite(relay1Pin, LOW);
  digitalWrite(relay2Pin, LOW);
  delay(initalTimeDelay);

  digitalWrite(5, LOW);
  digitalWrite(7, LOW);
  digitalWrite(12, HIGH);
  digitalWrite(13, HIGH);
  odorPrepareDone();
  digitalWrite(13, LOW);
```

```
    digitalWrite(12, LOW);
    digitalWrite(11, HIGH);
    digitalWrite(13, HIGH);
    delay(3000);
    odorPrepareDone();
    digitalWrite(13, LOW);
    digitalWrite(11, LOW);
    digitalWrite(9, HIGH);
    digitalWrite(13, HIGH);
    delay(3000);


}

//Odor prepare done
void odorPrepareDone() {
  buttonStatePrepare = digitalRead(buttonPinPrepare);
  while(buttonStatePrepare == LOW) {
    delay(prepareTimeDelay);
    buttonStatePrepare = digitalRead(buttonPinPrepare);
  }
}

//Since distance from odor cup to training tube
//compare to maunual system is too long, fill some distance odor
void odorFill(){
  digitalWrite(9, LOW);
  digitalWrite(13, LOW);
  digitalWrite(12, HIGH);    // turn the Powertail off
  delay(odorFillPeriod);

  digitalWrite(12, LOW);
  digitalWrite(9, HIGH);
  delay(odorFillPeriod);

  digitalWrite(9, LOW);
  digitalWrite(4, HIGH);
  delay(100);
  digitalWrite(4, LOW);
  delay(250);
  digitalWrite(11, HIGH);
  delay(odorFillPeriod);
  delay(odorFillPeriod);

  digitalWrite(13, HIGH);
}

void drosophilaPrepareDone() {
   buttonStatePrepare = digitalRead(buttonPinPrepare);
  while(buttonStatePrepare == LOW) {
    delay(prepareTimeDelay);
    buttonStatePrepare = digitalRead(buttonPinPrepare);
  }
}


//One cycle for training the drosophila
void oneCycleTraining() {
  //1=
  digitalWrite(13, LOW);
  //digitalWrite(11, LOW);
  for(int i=0; i<120; i++)
  {
    delay(1000);
  }
  digitalWrite(11, LOW);
  digitalWrite(9, HIGH);
  delay(3000);
  for(int shockCycle=1; shockCycle<=10; shockCycle++) {
    eightyShockControl();
  }

  //2
  digitalWrite(9, LOW);
  digitalWrite(11, HIGH);
  for(int i=0; i<60; i++) {
    delay(1000);
  }

  //3
  digitalWrite(11, LOW);
  digitalWrite(12  , HIGH);
  for(int i=0; i<60; i++) {
    delay(1000);
  }
  digitalWrite(12, LOW);
  digitalWrite(11, HIGH);
  for(int i=0; i<60; i++) {
    delay(1000);
  }
}

//Pin 13 for control the 80v shock
void eightyShockControl() {
  digitalWrite(10, HIGH);
  delay(2500);
  digitalWrite(10, LOW);
  delay(2500);
}

//15 minute air blank in between each cycle
void fifteenAirBlank() {
  digitalWrite(9, LOW);
  digitalWrite(11, HIGH);
  for(int i=1; i<=900; i++) {
    delay(1000);
  }
}

//Moving drosophila from training tube to resting tube
void moveTrainingToResting() {
  digitalWrite(11, LOW);
   //2
  CurrentPosition analogRead(sensorPin);
  goalPosition = 500;

  while(goalPosition > CurrentPosition) {
    digitalWrite(relay1Pin, HIGH);
    digitalWrite(relay2Pin, LOW);
    CurrentPosition = analogRead(sensorPin);
  }
  while(goalPosition < CurrentPosition) {
    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, HIGH);
    CurrentPosition = analogRead(sensorPin);
  }

    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, LOW);

    digitalWrite(4, HIGH);
    delay(100);
    digitalWrite(4, LOW);
    delay(250);
    digitalWrite(4, HIGH);
    delay(2000);


}

//Move from training tube to the center
void moveTrainingToCenter() {
  digitalWrite(11, LOW);
  //2
  CurrentPosition = analogRead(sensorPin);
  goalPosition = 500  ;

  while(goalPosition > CurrentPosition) {
    digitalWrite(relay1Pin, HIGH);
    digitalWrite(relay2Pin, LOW);
    CurrentPosition = analogRead(sensorPin);
  }
  while(goalPosition < CurrentPosition) {
    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, HIGH);
    CurrentPosition = analogRead(sensorPin);
  }

    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, LOW);

    delay(2000);

    digitalWrite(4, HIGH);
    delay(100);
    digitalWrite(4, LOW);
    delay(250);
    digitalWrite(4, HIGH);
    delay(2000);
}

//Stay center for learning training
void stayCenterPeriodForLearning() {
 //3

  CurrentPosition = analogRead(sensorPin);
  goalPosition = 680;

  while(goalPosition > CurrentPosition) {
    digitalWrite(relay1Pin, HIGH);
    digitalWrite(relay2Pin, LOW);
    CurrentPosition = analogRead(sensorPin);
  }
  while(goalPosition < CurrentPosition) {
    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, HIGH);
    CurrentPosition = analogRead(sensorPin);
  }

    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, LOW);
    delay(2000);
    digitalWrite(4, LOW);
    for(int i=1;i<= stayCenterTimeDelay ;i++) {
      delay(1000);
    }

}

//Resting in the tube
void restingPeriod() {
  //3
  CurrentPosition = analogRead(sensorPin);
  goalPosition = 540;
  while(goalPosition > CurrentPosition) {
    digitalWrite(relay1Pin, HIGH);
    digitalWrite(relay2Pin, LOW);
    CurrentPosition = analogRead(sensorPin);
  }
  while(goalPosition < CurrentPosition) {
    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, HIGH);
    CurrentPosition = analogRead(sensorPin);
  }

    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, LOW);
    digitalWrite(4, LOW);
  for(int i=1; i<=86400; i++) {
    delay(1000);
  }
}

//move drosophila from resting tube to the 3d inner part
void moveRestingToCenter() {

    digitalWrite(4, HIGH);
    delay(100);
    digitalWrite(4, LOW);
    delay(250);
    digitalWrite(4, HIGH);
    delay(2000);
}

//Stay in center period
void stayCenterPeriod() {
  //4
  CurrentPosition = analogRead(sensorPin);
  goalPosition = 810;

  while(goalPosition > CurrentPosition) {
    digitalWrite(relay1Pin, HIGH);
    digitalWrite(relay2Pin, LOW);
    CurrentPosition = analogRead(sensorPin);
  }
  while(goalPosition < CurrentPosition) {
    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, HIGH);
    CurrentPosition = analogRead(sensorPin);
  }

    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, LOW);
```

```
    delay(stayCenterTimeDelay);
}

//Drosophila choosing time
void choosePeriod() {
  digitalWrite(7, HIGH);
  digitalWrite(5, HIGH);
  //4
  CurrentPosition = analogRead(sensorPin);
  goalPosition = 810;

  while(goalPosition > CurrentPosition) {
    digitalWrite(relay1Pin, HIGH);
    digitalWrite(relay2Pin, LOW);
    CurrentPosition = analogRead(sensorPin);
  }
  while(goalPosition < CurrentPosition) {
    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, HIGH);
    CurrentPosition = analogRead(sensorPin);
  }

    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, LOW);

  for(int i=1;i<=chooseTimeDelay;i++) {
   delay(1000);
  }
}

//Stop choose period
void stopChoosePeriod() {
  //5
  CurrentPosition = analogRead(sensorPin);
  goalPosition = 640;

  while(goalPosition > CurrentPosition) {
    digitalWrite(relay1Pin, HIGH);
    digitalWrite(relay2Pin, LOW);
    CurrentPosition = analogRead(sensorPin);
  }
  while(goalPosition < CurrentPosition) {
    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, HIGH);
    CurrentPosition = analogRead(sensorPin);
  }

    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, LOW);

  digitalWrite(7, LOW);
  digitalWrite(5, LOW);
  digitalWrite(13, HIGH);

  odorPrepareDone();


  //6
  CurrentPosition = analogRead(sensorPin);
  goalPosition = 800;

  while(goalPosition > CurrentPosition) {
    digitalWrite(relay1Pin, HIGH);
    digitalWrite(relay2Pin, LOW);
    CurrentPosition = analogRead(sensorPin);
  }
  while(goalPosition < CurrentPosition) {
    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, HIGH);
    CurrentPosition = analogRead(sensorPin);
  }

    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, LOW);

  delay(2000);

  while(1 == 1) {
    delay(300000);
  }
}
```

Arduino Uno Data Logger Unit Code Notes of Both RATTS and LATTS

Since both RATTS and LATTS are data logging the same electronics, the code notes for data logging will be same for both RATTS and LATTS. The application code uploaded to the Arduino Uno data logger unit is detailed below:

```
#include <SD.h>
#include <Wire.h>
#include "RTClib.h"
#include <SPI.h>

//How many millisenconds between grabbing data and logging it. 500 ms
//is 0.5 second mills between entries
#define LOG_INTERVAL  500
//Mills between calls to flush() - to write data to the card
#define SYNC_INTERVAL 500
```

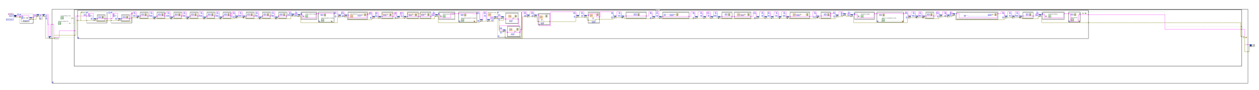
```
uint32_t syncTime = 0;//time of last sync()

#define ECHO_TO_SERIAL   1 // echo data to serial port
#define WAIT_TO_START    0 // Wait for serial input in setup()

//80v shock, gas sensor, and flow rate data logging input pins number
const int voltPin = A0; //Select input pin for 80v shock
int gasSensor1 = A1; // Select input pin for gasSensor
int flowRate1 = A2;
int flowRate2 = A3;
int flowRate3 = A4;

float sensorVal1 = 0; //Variable to store the value coming from the sensor
int flowVal1 = 0;
int flowVal2 = 0;
int flowVal3 = 0;
//80v voltage calculate
float denominator;
float resistor1 = 100600;
float resistor2 = 3287;

RTC_DS1307 RTC; // Define the Real Time Clock object

// For the data logging shield, we use digital pin 10 for the SD cs line
const int chipSelect = 10;

//The logging file
File logfile;
void error(char *str)
{
  Serial.print("error: ");
  Serial.println(str);
  while(1);
}

//Initialize all the input and output pins
void setup()
{
  Serial.begin(9600);
  Serial.println();
 // Serial.print("Initializing SD card...");
  pinMode(10, OUTPUT);
  //See if the card is present and can be initialized:
 if (!SD.begin(chipSelect)) {
    error("Card failed, or not present");
  }
  Serial.println("card initialized.");
  //Create a new file
  char filename[] = "LOGGER00.CSV";
  for (uint8_t i = 0; i < 100; i++) {
    filename[6] = i/10 + '0';
    filename[7] = i%10 + '0';
    if (! SD.exists(filename)) {
      //Only open a new file if it doesn't exist
      logfile = SD.open(filename, FILE_WRITE);
      break;  // leave the loop!
    }
  }
    if (! logfile) {
      error("couldnt create file");
  }
  Serial.print("Logging to: ");
  Serial.println(filename);
//Connect to RTC
  Wire.begin();
  if (!RTC.begin()) {
    logfile.println("RTC failed");
  #if ECHO_TO_SERIAL
    Serial.println("RTC failed");
  #endif  //ECHO_TO_SERIAL
 }
    logfile.println("millis,stamp,datetime,vccShock,odorConcentration1,odorConcentration2,flowRate");
  #if ECHO_TO_SERIAL
    Serial.println("millis,stamp,datetime,vccShock,odorSensor1,odorSensor2,flowRate");
  #endif //ECHO_TO_SERIAL
  denominator = (float)resistor2 / (resistor1 + resistor2);
}


void loop() {
  DateTime now;

//Delay for the amount of time we want between readings
  delay((LOG_INTERVAL -1) - (millis() % LOG_INTERVAL));

//Log milliseconds since starting
  uint32_t m = millis();
  logfile.print(m);           // milliseconds since start
  logfile.print(", ");
#if ECHO_TO_SERIAL
  Serial.print(m);         // milliseconds since start
  Serial.print(", ");
#endif

//Fetch the time
  now = RTC.now();
//Log time
  logfile.print(now.unixtime()); // seconds since 1/1/1970
  logfile.print(", ");
  logfile.print("");
  logfile.print(now.year(), DEC);
  logfile.print("/");
  logfile.print(now.month(), DEC);
  logfile.print("/");
  logfile.print(now.day(), DEC);
  logfile.print(" ");
  logfile.print(now.hour(), DEC);
  logfile.print(":");
  logfile.print(now.minute(), DEC);
  logfile.print(":");
  logfile.print(now.second(), DEC);
  logfile.print("");
#if ECHO_TO_SERIAL
  Serial.print(now.unixtime()); // seconds since 1/1/1970
  Serial.print(", ");
  Serial.print("");
  Serial.print(now.year(), DEC);
  Serial.print("/");
  Serial.print(now.month(), DEC);
  Serial.print("/");
  Serial.print(now.day(), DEC);
  Serial.print(" ");
  Serial.print(now.hour(), DEC);
  Serial.print(":");
  Serial.print(now.minute(), DEC);
  Serial.print(":");
  Serial.print(now.second(), DEC);
  Serial.print("");
#endif //ECHO_TO_SERIAL
  float voltage;
//Obtain RAW voltage data
  voltage = analogRead(voltPin);
//Convert to actual voltage (0 - 5 Vdc)
  voltage = (voltage / 1024) * 5.0;
  //Convert to voltage before divider
  //  Divide by divider = multiply
  //  Divide by 1/5 = multiply by 5
  voltage = voltage / denominator;
  //Output to serial
  logfile.print(", ");
  logfile.print(voltage);
  Serial.print(", ");
  Serial.print(voltage);

//Gas sensor1 data logger
  sensorVal1 = analogRead(gasSensor1); // read the value from the pot
  sensorVal1 = (sensorVal1/1024) * 5;
  logfile.print(", ");
  logfile.print(sensorVal1);
  Serial.print(", ");
  Serial.print(sensorVal1);

//Flow rate data logger
```

```
  flowVal1 = analogRead(flowRate1); // read the value from the pot
  flowVal1 = (flowVal1/1024) * 5;
  logfile.print(", ");
  logfile.print(flowVal1);
  Serial.print(", ");
  Serial.print(flowVal1);


//Flow rate data logger
  flowVal2 = analogRead(flowRate2);
  flowVal2 = (flowVal2/1024) * 5;
  logfile.print(", ");
  logfile.print(flowVal2);
  Serial.print(", ");
  Serial.print(flowVal2);


//Flow rate data logger
  flowVal3 = analogRead(flowRate3);
  flowVal3 = (flowVal3/1024) * 5;
  logfile.print(", ");
  logfile.print(flowVal3);
  Serial.print(", ");
  Serial.print(flowVal3);

  logfile.println();

#if ECHO_TO_SERIAL
  Serial.println();
#endif // ECHO_TO_SERIAL
  //Delay to make serial out readable
    if ((millis() - syncTime) < SYNC_INTERVAL) return;
  syncTime = millis();

  // blink LED to show we are syncing data to the card & updating FAT!

  logfile.flush();

} // void loop close
```

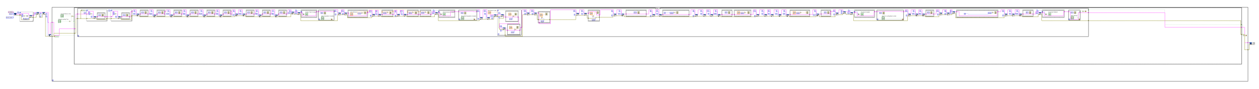
APPENDIX C

Arduino Uno Control Unit Block Diagram of RATTS

RATTS Learning Arduino Uno Control Unit Block Diagram



RATTS Memory Arduino Uno Control Unit Block Diagram



REFERENCES

[1] B. R. Malik and J. J. L. Hodge, “Drosophila adult olfactory shock learning.,” *J. Vis. Exp.*, no. 90, p. e50107, Jan. 2014.

[2] W. G. Tully, T., Quinn, “Classical conditioning and retention in normal and mutant Drosophila melanogaster,” *J. Comp. Physiol.*, 1985.

[3] R. L. Davis, “Olfactory memory formation in Drosophila: from molecular to systems neuroscience.,” *Annu. Rev. Neurosci.*, vol. 28, pp. 275–302, Jan. 2005.

[4] S. Murakami, C. Dan, B. Zagaeski, Y. Maeyama, S. Kunes, and T. Tabata, “Optimizing Drosophila olfactory learning with a semi-automated training device.,” *J. Neurosci. Methods*, vol. 188, no. 2, pp. 195–204, May 2010.

[5] I. Sram, S. P. W. M. Channels, O. A. Comparator, S. Sleep, M. Idle, and A. D. C. N. Reduction, “Features • Advanced RISC Architecture – 131 Powerful Instructions – Most Single Clock Cycle Execution – 32 x 8 General Purpose Working Registers – Up to 20 MIPS Throughput at 20MHz – Optional Boot Code Section with Independent Lock Bits In-System Programm.”

[6] SMC, “Compact Direct Operated 2/3 Port Solenoid Valve for Water and Air.” [Online]. Available: https://stevenengineering.com/tech_support/PDFs/70PVVDW.pdf.

[7] M. Margolis, *Arduino Cookbook*. Sebastopol, CA: O’Reilly, 2011.

[8] F. Semiconductor, “Tip100/tip101/tip102,” 2008. [Online]. Available: https://www.fairchildsemi.com/datasheets/TI/TIP102.pdf.

[9] “Specification of Product FEETECH RC Model Co ., Ltd . Specification of Product.” [Online]. Available: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Robotics/FS5106B specs.pdf.

[10] “Linear Actuator Specifications.” [Online]. Available: http://www.trossenrobotics.com/shared/productdocs/Linear-actuator.pdf.

[11] “Pololu - Concentric LACT6P-12V-20 Linear Actuator with Feedback: 6‘ Stroke, 12V, 0.5’/s.” [Online]. Available: http://www.pololu.com/product/2307. [Accessed: 04-Nov-2014].

[12] “RobotGeek Relay - An Arduino Compatible Relay Board.” [Online]. Available: http://www.robotgeek.com/robotgeek-relay. [Accessed: 05-Nov-2014].

[13] G. Technologies, “Stimulators - SD9 Square Pulse.” [Online]. Available: http://www.grasstechnologies.com/products/stimulators/stimsd9.html. [Accessed: 09-Nov-2014].

[14] Figaro, “TGS 2620 - for the detection of Solvent Vapors.” [Online]. Available: http://www.figarosensor.com/products/2620pdf.pdf. [Accessed: 05-Nov-2014].

[15] W. Eugster and G. W. Kling, “Performance of a low-cost methane sensor for ambient concentration measurements in preliminary studies,” *Atmos. Meas. Tech.*, vol. 5, no. 8, pp. 1925–1934, Aug. 2012.

[16] Figaro, “Technical Information on Usage of TGS Sensors for Toxic and Explosive Gas Leak Detectors.” [Online]. Available: http://www.figarosensor.com/products/common(1104).pdf. [Accessed: 05-Nov-2014].

[17] I. S. Technology, “Flow Sens FS5 Thermal Mass Flow Sensor for all-purpose use in Gases.” [Online]. Available: http://www.farnell.com/datasheets/484569.pdf. [Accessed: 05-Nov-2014].

[18] Arduino, “Arduino Playground - ThermalAnemometer.” [Online]. Available: http://playground.arduino.cc/Main/ThermalAnemometer. [Accessed: 06-Nov-2014].

[19] T. Instrument, “LM741 Operational Amplifier.” [Online]. Available: http://www.ti.com/lit/ds/symlink/lm741.pdf. [Accessed: 06-Nov-2014].

[20] S.-T. MICROELECTRONICS, “2N2222 Datasheet.” [Online]. Available: http://media.digikey.com/pdf/Data Sheets/ST Microelectronics PDFS/2N2222.pdf. [Accessed: 06-Nov-2014].

[21] T. Instrument, “LM340-N/LM78XX Series 3-Terminal Positive Regulators.” [Online]. Available: http://www.ti.com/lit/ds/symlink/lm340-n.pdf. [Accessed: 07-Nov-2014].

[22] L. Technology, “LT1635 Micropower Rail-to-Rail Op Amp and Reference.” [Online]. Available: http://cds.linear.com/docs/en/datasheet/1635fa.pdf. [Accessed: 07-Nov-2014].

[23] W. E. Last and P. M. Edt, “Guide Contents Overview The new and improved logging shield Installing the Headers Assembly with male headers Assembly with Stacking Headers : Shield Overview Using the Real Time Clock Using the SD Card For Mega and Leonardo Users ! For the Mega and Leona,” 2014. [Online]. Available: http://learn.adafruit.com/downloads/pdf/adafruit-data-logger-shield.pdf.

[24] Stratasys, “Dimension BST 768, SST 768, & Elite User Guide.” [Online]. Available: http://fab.cba.mit.edu/content/tools/dimension/Dimension 768 Elite User Guide.pdf. [Accessed: 09-Nov-2014].

[25] PTC, “PTC Creo Parametric.” [Online]. Available: http://support.ptc.com/WCMS/files/161682/en/PTC_Creo_Parametric_Data_Sheet.pdf. [Accessed: 09-Nov-2014].

[26] S. B.-A. Products, “Bel-Art Riteflow Panel/Bench Mounted Flowmeters, SCIENCEWARE H404060020 65 Mm Flowmeters With Aluminum Fittings. Bel-Art Labware & Accessories.” [Online]. Available: http://www.opticsplanet.com/bel-art-riteflow-panel-bench-mounted-flowmeters-scienceware-h404060020-65-mm-flowmete.html. [Accessed: 14-Nov-2014].