INSIGHT MAZE: An Open Source,

Automated Rodent Maze

USER GUIDE

Gregory Suematsu

Gina Poe PhD

Bioengr 199

**TABLE OF CONTENTS**

**Doors…………………………………………………………………………………………...2**

Hardware……………………………………………………………………………….2

Software………………………………………………………………………………..2

**Sensors………………………………………………………………………………………...3**

Hardware……………………………………………………………………………….3

Software………………………………………………………………………………..3

**Reward Dispensers…………………………………………………………………………..5**

Hardware……………………………………………………………………………….5

Software………………………………………………………………………………..6

**Master…………………………………………………………………………………………..6**

Hardware……………………………………………………………………………….6

Software………………………………………………………………………………..6

**Figures…………………………………………………………………………………………9**

1. **Doors**
   1. **Hardware**

The doors for the maze are controlled by servo motors that rotate a 3D printed part which will raise the door. The printed part, or door arm, is screwed into the servo horn and rotates with the horn of the servo to the specific angular position in the code. The approximate length of the door arm is 7”, and the servo will rotate about 100 degrees between the open and the closed states. This means that the servo with the door arm is able to lift the door over 10”, the height of the doors. The tradeoff between the length of the door arms is that the shorter arms will occupy less space and will not be able to lift the doors to their full height. However, the longer the door arm, more torque is required to lift the doors. The model of the door mechanism and set up are in Figure 1.

The servos require a separate power supply for each module because they require a high current supply, which the Arduino nano controlling each module cannot supply. There is a rail of the pcb of each module that is connected to the power supply that is plugged into the wall. Additionally, there will be a new power supply that will replace the 9V battery to power each Arduino nano. These power supplies will have a barrel connector on one end, which is a coaxial connecter. When you completely cut off the end, two wires conductors are exposed; the black one is generally ground, and the red one is generally the positive voltage. If you are unsure, measure the voltage with a multimeter, but be careful to not touch the conductors together or to not touch them at the same time. In regards to the wiring of each servo, the **brown wire goes to ground**, the **yellow goes to the door’s specific pin**, and the **orange goes to the +5V** from the power supply. The URLs to the parts are below.

Servo: <https://www.adafruit.com/product/1142>

Servo Power Supply: <https://www.adafruit.com/product/658>

Suggested Nano Power Supply: <https://www.adafruit.com/product/352>

The servo horns will have a door arm screwed into each one; however, installing each horn must be specific. All the doors should be put into the open state using the software code. Then the arms can be screwed on. This ensures that the door arm is oriented in the correct direction, there may be doors whose open and closed positions in the code should be changed in the software, but should try to avoid this.

* 1. **Software**  
      The software to control the doors in each module are found in the “InsightMaze.h” file. Most of the functions call functions of the door class, but in order to control the doors of each module, only the module functions should be used.
     1. void openDoor(int doorID) --- Opens the door with specific door ID
     2. void closeDoor(int doorID) --- Closes the door with specific door ID
     3. bool isDoorOpen(int doorID) --- Returns whether or not the door with specific door ID is open - if the specified door is open the function will return a 1 or true, otherwise it will return 0 or false.
     4. void openAllDoors() --- Opens all the doors of the module
     5. void closeAllDoors() --- Closes all the doors of the module
     6. void testDoors(int closeTime, int openTime) --- Closes all the doors of the module, waits closeTime in milliseconds, then opens all the doors, waits openTime milliseconds
     7. void printDoorsStates() -- Prints out the states of all the doors in the module - if the door is open the function will return a 1 or true, otherwise it will return 0 or false.
     8. void setPath(int path) -- Configures the doors to a specific path of the maze - takes arguments of the following: LEFT, RIGHT, CENTER

Within the “globals.h” file, there are constant values that are used for the doors. Specifically, the door states and pins. The door pins are the pins that control each door (Ex. PIN\_DOOR\_0), and the door states (Ex. POS\_D0[]) are the angular position the servo goes to be in the open or closed state. The first value of the array is the open position and the second value of the array is the closed position. These values can be changed module to module. I recommend keeping track of each set of door positions for the specific module by commenting in the code which set of positions are associated with which module.

1. **Sensors**
   1. **Hardware**

The hardware for each sensor consists of an infrared LED and an infrared phototransistor. The infrared LED emits IR light away from the sensor, and any object in front of the sensor will reflect the light back. The closer the object, the more IR light the phototransistor will detect. When more IR light is detected by the phototransistor, more current is allowed to flow, which causes a larger voltage drop across the resistor, and thus a smaller voltage reading by the Arduino. The circuit used is shown in Figure 2. The **white wire goes to the analog read pin** for the specific sensor, the **brown wire goes to ground**, and the **orange wire goes to the +5V source** of the Arduino (Do not confuse this with the +5V from the power supply for the doors). The IR sensors were purchased from Adafruit, and the URL to the website is below.

Infrared Sensor: <https://www.adafruit.com/product/3930>

* 1. **Software**

The software for the sensors is not straight forward. Recall, when there is an object in front of the sensor, the Arduino will read a smaller voltage. Ideally, the change in the value between an object in front and nothing in front is about 200-300 bins. Bins are the units for the returned value by the analogRead() function of Arduino. Bins correlate to the analog to digital converter, and to convert these values to voltages the following equation is used.

The drop in voltage is how the Arduino know there is an object in front of it. However, the Arduino must be told to read the voltage of the pin to know its value. When the module is turned on, the sensors take an initial reading and calculate their threshold value based on the ambient light. Most events triggered by a sensor occur when an object leaves the view of the sensor, which is referred to as a falling edge. This may be counterintuitive because the values of the sensor actually rise when an object leaves the sensor’s view, but the falling edge refers to the question/boolean “Is an object in view of the sensor?”. If an object is in view of the sensor, the question is answered “yes” or “true” or “1”. When the object leaves the view, the answer becomes “no” or “false” or “0”. The falling edge refers to the change from “1” to “0”. The functions to use the sensors are below.

1. int getSensorVal(int sensorID) --- Returns the sensor value (0-1023) of the sensor with specific sensor ID
2. void updateSensors() -- ***THIS IS THE MOST IMPORTANT SENSOR FUNCTION***. Must be called at the beginning of void loop() to update the sensors values and states. If this is not used, falling and rising edges will not be detected.
3. bool isSensorFall(int sensorID) -- Returns whether or not an object left the view of the sensor during the loop
4. bool isSensorRise(int sensorID) --- Returns whether or not an object entered the view of the sensor during the loop
5. int getSensorThresh(int sensorID) --- Returns the threshold value of the sensor with the specific sensor ID
6. bool isObjInSensorView(int sensorID) --- Returns whether or not an object is in the view of the sensor
7. bool wasObjInSensorView(int sensorID) --- Returns whether or not an object was just in the view of the sensor during the past loop
8. void printSensorsVals() --- Prints all the values of the sensors of the module
9. void printSensorsThresh() --- Prints the thresholds of all the sensors of the module
10. void printIsObjSensorView() -- Prints whether an object is in view of all sensors of the module
11. void printWasObjSensorView() --- Prints whether an object was just in view of all sensors of the module
12. void printSensorsIsFall() --- Prints whether an object just left the view of all sensors of the module
13. void printSensorsIsRise() --- Prints whether an object just entered the view of all sensors of the module
14. void testSensors(int values) --- Prints specified values of the sensors in the module - int values should take the format of 0b00000 - there will only 6 values of each sensor you will print. If you want to print out more than one of these values, it is recommended you use this function. Each 0 in the “values” parameter corresponds to each of the sensor variables.
    1. Bit 0 - printSensorsVals()
    2. Bit 1 - printSensorsThresh();
    3. Bit 2 - printIsObjSensorView();
    4. Bit 3 - printWasObjSensorView();
    5. Bit 4 - printSensorsIsFall();
    6. Bit 5 - printSensorsIsRise();

For example, if the function was called in the following manner void testSensors(0b011100);, printIsObjSensorView(), printWasObjSensorView(), and printSensorsIsFall() would all be called.

If the sensors malfunction or the voltage difference between the states changes but is still consistent, the voltage difference value (SENSOR\_THRESHOLD\_DIFF) can be changed in the “Sensor.h” file. Additionally, the pins the white wires are plugged into are shown in “globals.h” as constant values (Ex. PIN\_SENSOR\_0[]).

1. **Rewards Dispensers**
   1. **Hardware**

The rewards dispensers are modified syringe pumps. The stepper motor pushes on the plunger of the syringe to dispense the reward. The stepper motor is driven using +12V power supply. The rewards dispenser design relies on the fact that the stepper rotates a threaded shaft that can rotate freely, but when actuated and prevented from rotating, the motion to force to be linear. There are removable side walls on each dispenser that can be removed to rotate the shaft and platform to press against the plunger. When in use, the side walls should be in place, so when the motor is actuated, the platform pushes the syringe. Figure 3 shows a model of the rewards dispenser and labeled image of the stepper driver. The URLs to the parts are below.

Stepper Driver: <https://www.sparkfun.com/products/12779>

Stepper Motor: <https://www.sparkfun.com/products/10848>

Power Supply: [https://www.mouser.com/ProductDetail/MEAN-WELL/RD-85B](https://www.mouser.com/ProductDetail/MEAN-WELL/RD-85B?qs=l0g2inPJSHPh2l1bLnrx1w%3D%3D)

* 1. **Software**

The software for the syringe pump is very simple. There is only one command, which is to have a specific reward dispenser dispense a specific volume of reward. The step and direction pins for each rewards dispenser are determined in “globals.h”, for example, SYRINGE\_0\_STEP\_PIN.

* + 1. void dispenseReward(int syringePumpID, float vol) --- Tells syringe pump of specific ID to dispense specified volume in milliliters. This is the only function for the syringe pump. Any other movements with the syringe pump are done by hand.

1. **Master**
   1. **Hardware**

The master consists of a Arduino Mega 2560 and a customized shield. The shield consists of a button, SD card reader, module connections, a prototyping area, and alert pins out, shown in Figure 4.

The button is utilized by the user to advance the procedure of the maze. Each button press changes the state of the maze, and the state determines what the modules do during the time.

The SD card reader is used to read the text file “PATHS.TXT” on the SD card. The text file must be named this otherwise the master will not recognize it. The text file must their path in character form (Ex. “rll” for right-left-left) followed by a new line character. The master is able to store up to 60 paths but this can be changed in the code.

The specific module connections are labeled on the shield. Each label corresponds to the pin on the nano of the module it should be connected. The ID pin must be connected to the ID pin of the module, which can be changed in the code, through an electronic low pass filter. The electronic filter is a first order, passive RC circuit. The filter is used to average the PWM signals sent out by the master to each module as a way to identify them. The suggested R and C values are 1 MOhm and 1 uF, respectively. Any set up that filters out the PWM high frequency will suffice.

The prototyping area is used to make any modifications needed without making a new PCB. The alert pins are meant for interfacing with OpenEphys; each alert header has a GPIO pin along with a GND pin to send digital signals to other devices.

* 1. **Software**

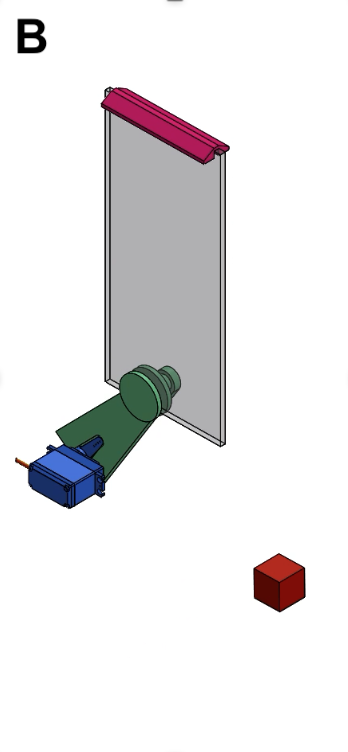
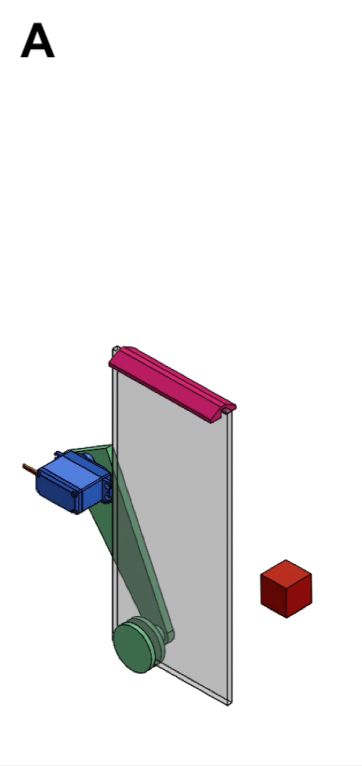
The software with the master maze interfaces with the modules using serial communication. The same set of commands are sent to every module, but the module only looks for the commands associated with its ID. The module’s ID is determined by the pin it is plugged into on the master. The pin that determines the ID of the module and specific is found in “InsightMaze.h” and “InsightMazeMaster.h”. Each pin sends out a respective PWM signal specified by ID\_VALS\_OUT[] in “InsightMazeMaster.h”.

The commands are formatted as string similar to the following: “a130b131c129d000f000”. The characters are the IDs of each module and the 8-bit integer followed by the character are the commands of the respective module. The useful functions are below.

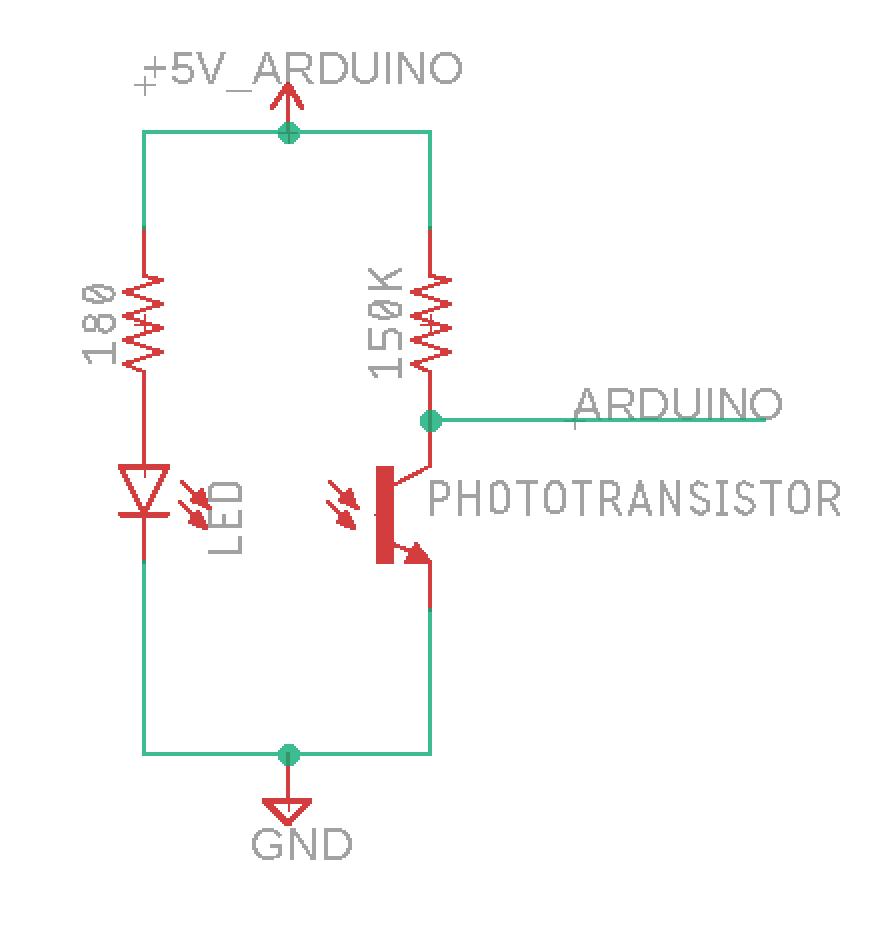
1. void setCommands(int commands[]) --- Takes array of length 6 to send to each module and stores as master's commands - the numbers in the commands array correspond to the commands for each module.
2. void transmitCommands() --- Transmits the master's stored commands, which are set during setCommands
3. void updateBtnVals() --- MUST have at the beginning of each loop to update button state - without this function, the button loses functionality.
4. bool isBtnPressed() --- Returns true if the button was pressed during the loop and updates m\_btnCounter, which keeps track of the total number of button presses
5. unsigned int btnCounter() --- Returns the number of button presses (returns the value of m\_btnCounter)
6. unsigned int state() --- Returns state based on the number of button presses - the state is used to determine what the master should be doing
7. void getPathsFromSD() --- Reads SD card and stores the paths in the master object - reads the paths from the text file “PATHS.TXT”.
8. void printPaths() --- Prints all the stored paths within the master object
9. String getPath(int pathNum) --- Returns the string of specified path number - the path number corresponds to the line the path was on in the text file. For example, if the path “lcl” was on the first line getPath(0) would return “lcl”
10. int\* sctSplitPathCommands(String paths, int hiddenRule) --- Creates array for specific path and hidden rule location that will be transmitted to the modules. The pointer to the first command is returned and is used to pass into setCommands.
11. void sctProtocol(int hiddenRule) --- Executes the protocol for SCT experiments - the int hiddenRule indicates which forced decision is the direction that will result in a reward in the final module - the function iterates through the paths for each button press.
12. void sendAlert() --- Sends pulse to alert out pin of master. Meant to be used as interrupt function to send signal to openEphys
13. int pinAlertIn() --- returns the alert pin input from the end module. Do not confuse this pin with the alert out pins on the shield - these are two different pins with separate functions.

The software for the code only needs to be edited so that an interface with openEphys is possible.

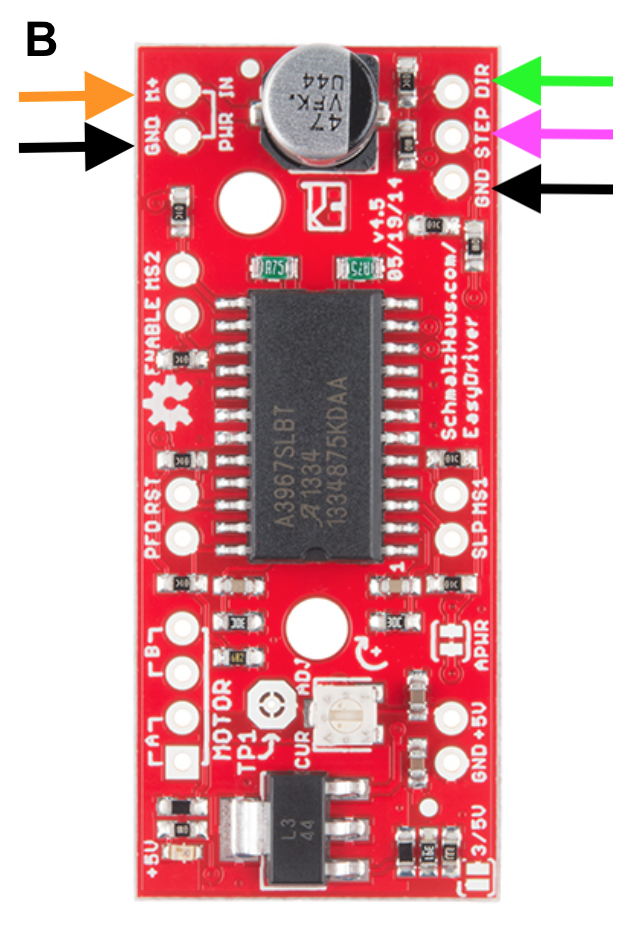
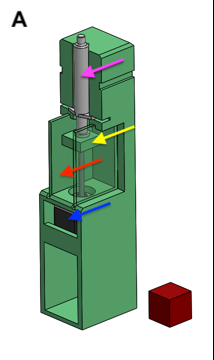
1. **Figures**

****

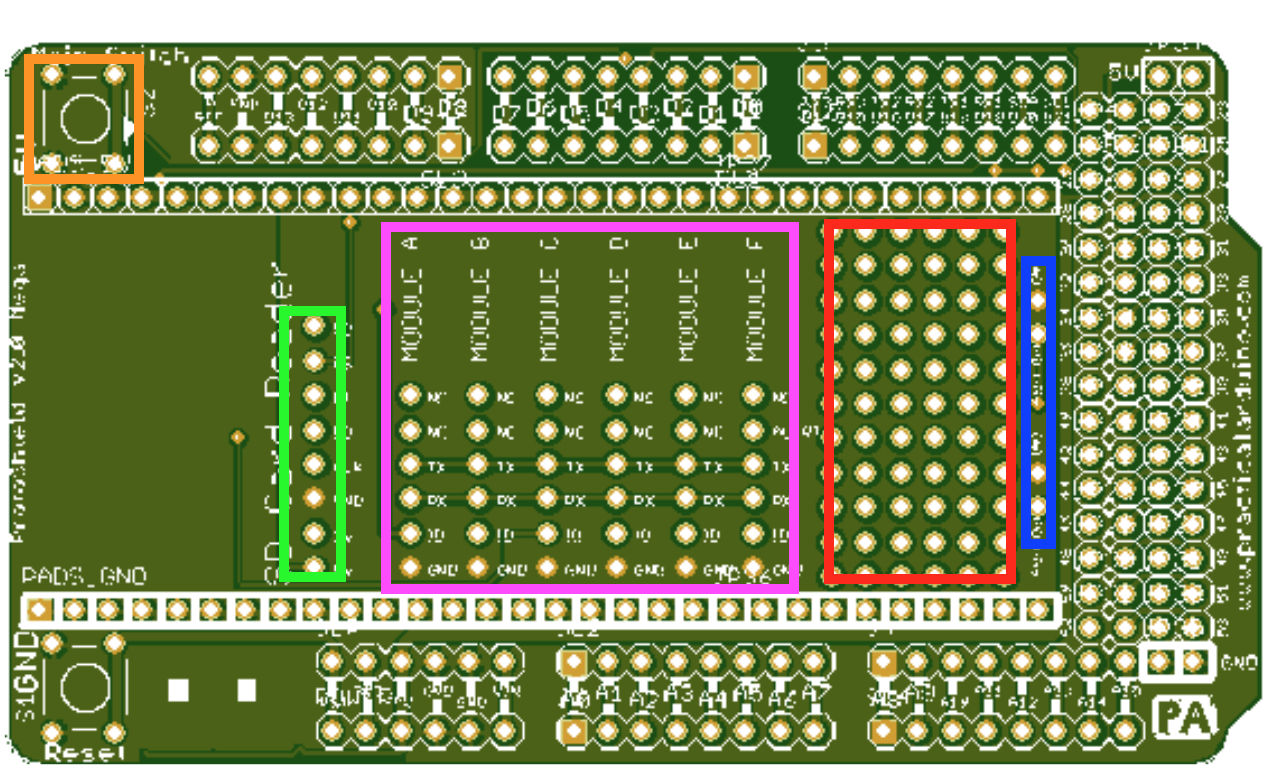
**Figure 1: Maze Door Mechanism.** The door arm (green), is attached to the servo (blue) using a screw in the horn. The servo rotates between two states: open (A) and closed (B). The door is shown as transparent in this image and the door cap to cover the slot the door slides between.The servo motor is mounted to a wood stand that is not pictured in the figure. The red cube is a 1” cube for scale.

****

**Figure 2: Sensor Circuit.** The part bought from Adafruit contains the LED and Phototransistor in the image. Both sides of the circuit share the same +5V source from the nano. The pins from each device (LED and phototransistor) that go to ground are soldered together and connected to a wire. The wire that goes to the analog read pins of the nano is connected between the phototransistor and its respective resistor. The soldered sensors are then insulated using hot glue.



**Figure 3: Rewards dispenser.** The stepper driver (B) is used to control the syringe pump (A) with the arduino. **(A)** The rewards module consists of side walls (red arrow), stepper motor (blue arrow), platform (yellow arrow), and syringe (pink arrow). The red cube is a 1” cube for scale. **(B)** The driver is oriented in this fashion on the rewards dispenser. The black arrows indicate where the ground wires are connected. The orange and black arrows on the left are connected to the +12V power supply. The black, pink, and green arrows on the right should be connected to the wires leading to the module’s arduino. The motor is connected to the bottom left, and the paired wires should be soldered together. Look as stepper motor documentation for this.



**Figure 4:** **Master Shield.** The shield consists of a button (orange), SD card reader (green), module connections (pink), a prototyping area (red), and alert pins out (blue). The prototyping area has vias were wire and parts can be soldered; the vias are connected in triplets from left to right.