

Configuring Bluetooth HC-05 in AT mode with Arduino Uno

The HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup [1]. In order to configure the module, it is necessary to enter into AT command mode and to transmit data the Bluetooth module will need to be out of AT command Mode.

AT mode refers to the form of communication to the HC-05 Bluetooth Module. AT Commands are short for Attention Commands which is a command language used for modems known as the Hayes command set. Hayes command set is a specific command language originally developed by Dennis Hayes for the Hayes Smartmodem 300 baud modem in 1981 [2]. The HC-05 Bluetooth Module was used due to its ability to be configured as Master or Slave mode as well as adding a password to the module. Initially for this project the HC-06 module was going to be used which is only in Slave Mode and it does not have the ability to be configured.

Parts used to configure are the following:

- 2 HC-05 Bluetooth Modules
- Arduino Uno
- Breadboard
- 4 wires
- USB A to B cable
- Computer
- open-source Arduino Software (IDE)

To enter AT Command mode using an Arduino Uno wired to a HC-05 Bluetooth Module one must first familiarized with the physical Bluetooth Module.

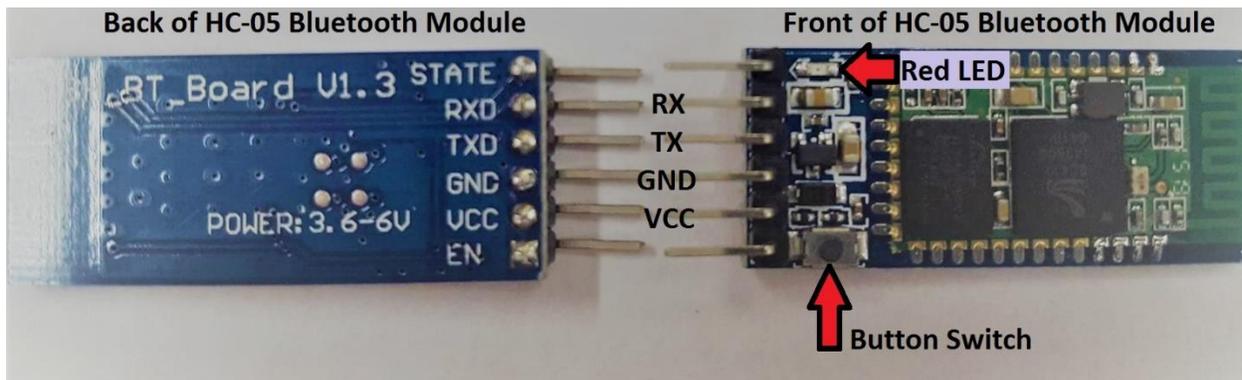
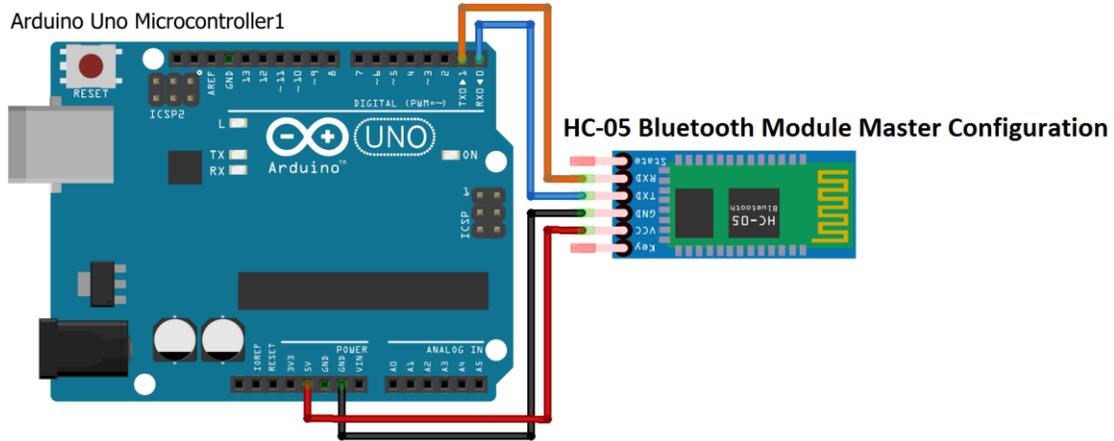


Figure 1. HC-05 Bluetooth Module

The Data sheet for the module can be found at: [https://www.itead.cc/wiki/Serial_Port_Bluetooth_Module_\(Master/Slave\)_:_HC-05](https://www.itead.cc/wiki/Serial_Port_Bluetooth_Module_(Master/Slave)_:_HC-05) which has a list of AT commands as well as specifications of the module. The Module needs a minimum of 3.3V to work and a maximum of 6V. The default Baud rate is 38400 and for this project out of the 6 pins that the module has, the outermost pins (STATE and EN) will not be used. The module can communicate up to 30 feet. As seen in Figure 1 the pins that will be used are RX, TX, GND, VCC. Additionally, from figure 1 the front of the has a red LED and a Button Switch.



The red LED has 3 continuous flashing modes: ON for 2 seconds and Off for 2 seconds (AT Command Mode), fast blinking (searching for a connection), Off for 2 seconds and blinks twice (connected). Figure 2 as well as Table 1 show the connections needed to configure the Bluetooth Module.

HC-05 Bluetooth Module Pin	Arduino Uno Pin
VCC	5V
GND	GND
TXD	TXD
RXD	RXD

Table 1. Connection between the pins of the Bluetooth Module and Arduino Uno

Figure 2. Entering AT Command Mode Schematic

To enter AT Command Mode the following must be done:

1. Arduino Uno must be connected to the Computer via USB and Arduino software opened
2. The Module must be disconnected from Arduino
3. Arduino needs to have a Blank Sketch downloaded

```
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

4. The Button Switch must be held pushed and simultaneously connected to the Arduino as seen in figure 2 (This is done most easily if using a breadboard)

- The Button Switch can be released once connected to Arduino and the Bluetooth Module LED should be blinking ON for 2 seconds and Off for 2 seconds indicating it has entered AT Command Mode
- The Correct COM Port should be Selected and Serial Monitor needs to be opened as seen in

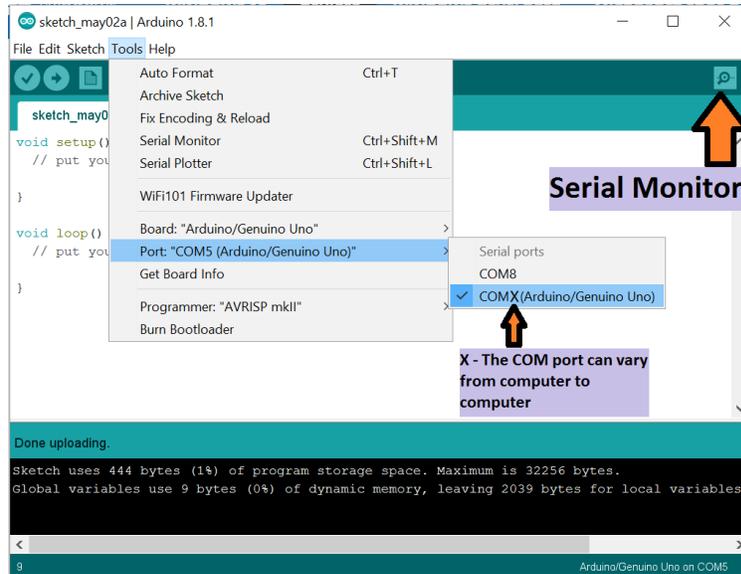


figure 3

- 38400 baud rate should be selected and "Both NL & CR" as seen in figure 4

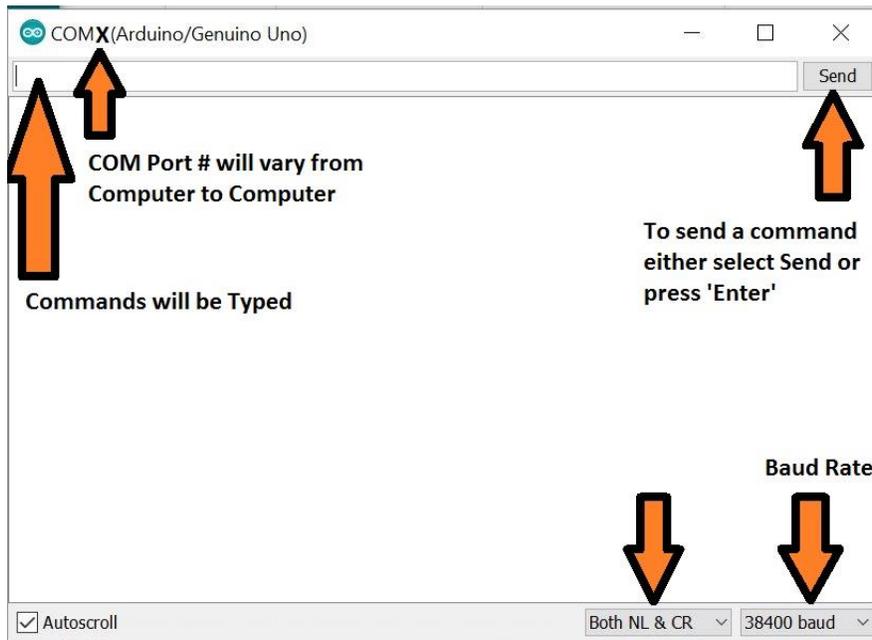


Figure 4. Arduino Serial Monitor

AT Command Mode

Using the commands found in reference [2] the following table 2 and table 3 are formed for Master and Slave Configuration and Table 4 and Table 5 are formed for fixed connection. Each Module is connected one at the time and must be in AT command Mode as explained above. The module is connected to the Arduino with a Blank Sketch and configure using the selected AT commands in tables 2,3,4,5. Figure 5 shows a screen capture of the Serial Monitor screen as a result of following Table 2 as an example of what to expect when using the Serial Monitor. **Commands have to be typed in capital letters**, when a command is typed it will disappear and a response from the module appears.

Commands Typed	Response from Module	Info
AT	Error: (0)	Make sure module in AT mode from Red LED blinking on 2 seconds and off 2 seconds
AT	OK	Communication Stablished
AT+ORGL	OK	Resets Module to factory settings Slave mode, pin code :1234, device name: H-C-2010-06-01 ,Baud 38400bits/s.
AT+UART=38400,0,0	OK	Setting Baud Rate
AT+UART?	+UART:38400,0,0 OK	Response of Module indicating set Baud Rate
AT+ADDR?	+ADDR:18:91:d8fff6 OK	Module Address Should be written for later use
AT+NAME=Latch-M	OK	Setting name for Module as Latch-M
AT+ROLE=1	OK	Setting Module as Master Mode
AT+ ROLE?	+ROLE:1 OK	Returns That it is in Master Mode
AT+PSWD=LATCH1	OK	Setting Password
AT+ PSWD?	+PSWD:LATCH1 OK	Returns that password is LATCH1 (Password must match the Slave Mode Module Password)

Table 2. Configuration used for the Master Mode Module

Commands Typed	Response from Module	Info
AT	Error: (0)	Make sure module in AT mode from Red LED blinking on 2 seconds and off 2 seconds
AT	OK	Communication Stablished
AT+ORGL	OK	Resets Module to factory settings Slave mode, pin code :1234, device name: H-C-2010-06-01 ,Baud 38400bits/s.
AT+UART=38400,0,0	OK	Setting Baud Rate
AT+UART?	+UART:38400,0,0 OK	Response of Module indicating set Baud Rate
AT+ADDR?	+ADDR: 18:91:d9198e OK	Module Address Should be written for later use
AT+NAME=Latch-S	OK	Setting name for Module as Latch-M
AT+ROLE=0	OK	Setting Module as Slave Mode

AT+ ROLE?	+ROLE:0 OK	Returns That it is in Slave Mode
AT+PSWD=LATCH1	OK	Setting Password
AT+ PSWD?	+PSWD:LATCH1 OK	Returns that password is LATCH1 (Password must match the Master Mode Module Password)

Table 3. Configuration used for the Slave Mode Module

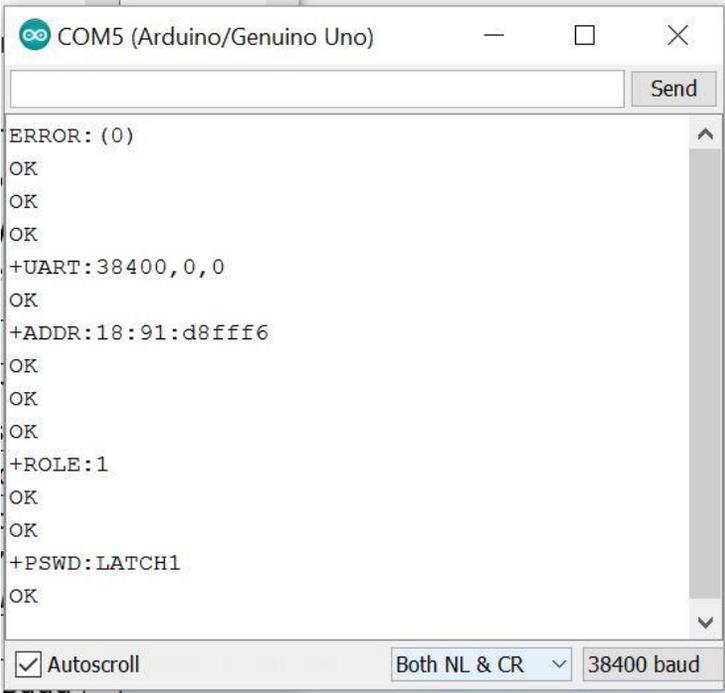


Figure 5. Screenshot of Serial Monitor of Configuring Master Mode Module

Table 2 and 3 will result in different address since each Bluetooth module is has a different address and changing the name and password is optional. Once one module is set as a Master and the other as a Slave the modules will connect automatically as long both are ON and within 30 feet assuming default password of 1234 is kept. If the password and name of each module are change the following Table 4 and Table 5 can be followed to set the modules to connect to each other with the new password and name.

Commands Typed	Response from Module	Info
AT	Error: (0)	Make sure module in AT mode from Red LED blinking on 2 seconds and off 2 seconds
AT	OK	Communication Stablished
AT+CMODE=0	OK	Set to connect to a fixed address
AT+CMODE?	+CMOD:0 OK	Confirms that module is set to connect to a fixed address

AT+BIND=18,91,d9198e	OK	Using the address from Slave mode module from Table 3 the fixed address to connect to is set (Notice that the address uses , instead of :)
AT+BIND?	+BIND:18:91:d9198e OK	Confirms that it is set to connect to the set address

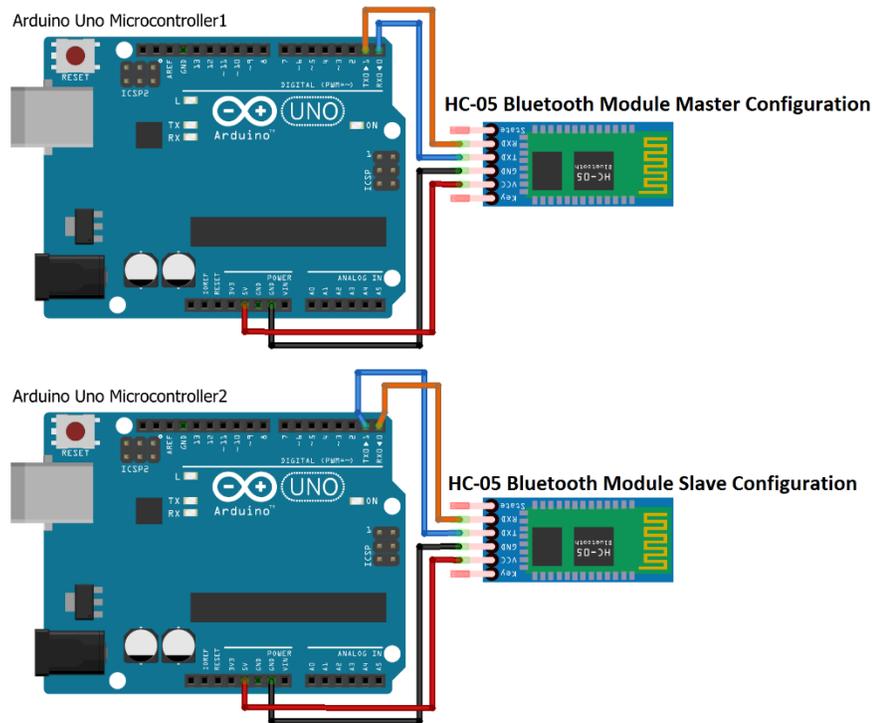
Table 4. Table for Master Mode Module Auto connect to Slave Mode Module

Commands Typed	Response from Module	Info
AT	Error: (0)	Make sure module in AT mode from Red LED blinking on 2 seconds and off 2 seconds
AT	OK	Communication Stablished
AT+CMODE=0	OK	Set to connect to a fixed address
AT+CMODE?	+CMOD:0 OK	Confirms that module is set to connect to a fixed address
AT+BIND=18,91,d8fff6	OK	Using the address from Slave mode module from Table 3 the fixed address to connect to is set (Notice that the address uses , instead of :)
AT+BIND?	+BIND: 18:91:d8fff6 OK	Confirms that it is set to connect to the set address

Table 5. Table for Slave Mode Module Auto connect to Master Mode Module

Connecting HC-05 Master Mode Module to a HC-05 Slave Mode Module

Once both Modules have been connected they can be wired to their own Arduino Uno Board with the Arduino Uno boards having the desired Code and start transmitting data. The Master Module is wired the same way that the module was wired in figure 2 to enter AT Command Mode. The Slave Mode Module will have the TX and RX wires switch as seen in Figure 6 and Table 6.



fritzing

Figure 6. Master and Slave Mode Module Wiring Configuration

HC-05 Bluetooth Master Mode Module Pins	Arduino Uno Master Board Pins	HC-05 Bluetooth Master Mode Module Pins	Arduino Uno Slave Mode Board Pins
VCC	5V	VCC	5V
GND	GND	GND	GND
TXD	TXD	TXD	RXD
RXD	RXD	RXD	TXD

Table 6. Pin connectors for HC-05 Bluetooth Module to Arduino Uno

When downloading code to Arduino Uno the HC-05 Module must be disconnected otherwise the download will not complete. **Unless the Arduino Uno will not be modified in the future the HC-05 Bluetooth Module should not be permeably connected.** A six pin female header as seen in figure 7

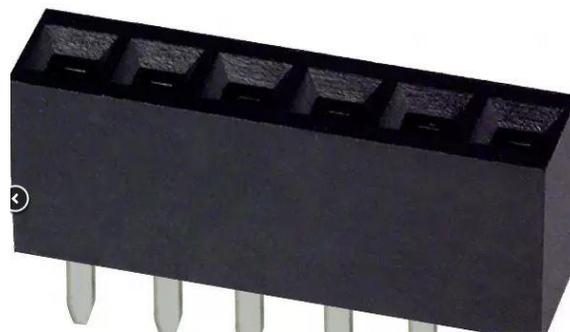


Figure 7. Example of 6-pin Female Header [3]

would be ideal solution for mounting Arduino so it can be removed when loading new code to Arduino Uno.

Troubleshooting HC-05 Bluetooth Module

Problem	Solution
Module red LED NOT blinking on for 2 seconds and off for 2 seconds, indicating that it is in AT Command Mode	<ul style="list-style-type: none"> • Remove power from System to check wires to make sure properly wired as seen in Figure 2. If properly connected apply power to system again • Disconnect Module and Reconnect while holding the Button Switch and releasing the Button Switch after it is Connected
Serial Monitor not communicating with HC-05 Bluetooth Module	<ul style="list-style-type: none"> • Make sure Arduino Uno has Blank code downloaded • Make sure red LED in module blinking 2 seconds and off for 2 seconds, indicating that it is in AT Command Mode • Make sure as seen in Figure 3 that the correct COM Port is selected • Make Sure Wired Correctly as seen in Figure 2
Code not downloading to Arduino Uno	<ul style="list-style-type: none"> • Make sure HC-05 Bluetooth is removed from the system while the code downloads • Make sure code does not have errors • Make Sure USB is properly connected to computer • Reset Arduino and try download again • Restart Arduino software and re-download code
HC-05 Bluetooth Module getting HOT	<ul style="list-style-type: none"> • Immediately but Safely remove power from system • Wait until module has cooled down because now Module is a Burning Hazard • After it has cooled Check wiring as seen in figure 6 • If module not turn On it will need to be replaced
Master and Slave Mode Bluetooth Modules not pairing	<ul style="list-style-type: none"> • Make sure Modules with in 30 feet of each other • Check power and connections as seen in figure 6 • Confirm that Master and Slave Modules are properly wired as seen in Figure 6 especially look at RXD and TXD pins

	<ul style="list-style-type: none">• Reset to factory settings and reconfigure Modules
--	---

Latch Mechanism Control Box Quick Overview

The Latch Mechanism Control Box will be powered from the orange power rails at the observatory. There is a 3 pin Female connector at the bottom of the control box labeled AC IN. On the Left hand side there is a 3 pin Female connector labeled AC OUT which controls the Dome Slit motor. At the top edge of the Control Box there is a 12pin connector which connects to the motors and Pressure Sensor as seen in the schematic of Figure 8.

The Control Box has a toggle switch in the front which can be moved upward or downward. If positioned upward towards the 'M' it will be in Manual mode. In manual mode the power will enter the bottom of the control box and directly exit to the output to the Slit Motor. If the toggle switch is positioned downward towards the 'A' power will enter the box and distributed inside the control box powering the 12V power supply. From the 12V power supply the Arduino is turned on along with the Motor shield which the Motors, Pressure Sensor and the Slave Mode Bluetooth Module are powered from as seen in the schematic of Figure 8.

In order for the Control box to Function The Arduino inside the Control Box will receive Data from the Master Mode Bluetooth Module. In the Test set up it was attached to an Arduino with push buttons sending the data through the Arduino as seen in Figure 9. Ideally in the future the data would be received from a Bluetooth module that is with the Main Control Box at the Observatory.

Coding

Here is an example code for using two HC-05 The following Code was found at <http://www.martyncurrey.com/connecting-2-arduinios-by-bluetooth-using-a-hc-05-and-a-hc-06-pair-bind-and-link/> [4], which uses a Master and Slave Bluetooth module. This Link was used to get started with the Code used for the Latch Mechanism Control Box.

The following components were used inside the Latch Mechanism Control Box where the Slave Mode Bluetooth was placed:

- Arduino Uno
- Adafruit V2 Motor Shield
- HC-05 Bluetooth Module (Configured as Slave Mode)
- PS-WP12LPS30 30W 12VDC Power Supply
- 1M ohm Resistor
- 2 5v SPST Relays
- 2 28V DPDT Relays

The following components is set up to be connected to the Latch Mechanism Control Box

- DC Motor-RS775-5 Motor With Encoder For PG71 and PG188 Gearbox with Bane Bots P60S-44-57 16:1 Gear Box attached
- Pressure Sensor-Flexiforce Pressure Sensor - 100lbs.
- Bipolar Stepper Motor-Stepping Motor Nema 17 Stepping Motor 26Ncm Step angle 1.8 Degrees

The following table 7 shows the pins that the 12 pin connector and what component it connects to externally.

Table of connections of the 12 Pin connector located on the top of the Latch Mechanism Control Box		
Pin	Connection on Motor Shield	Connection to Components
1	M3	Side A of Coil 1 of Bipolar Stepper Motor
2	M3	Side B of Coil 1 of Bipolar Stepper Motor
3	M4	Side A of Coil 2 of Bipolar Stepper Motor
4	M4	Side B of Coil 2 of Bipolar Stepper Motor
5	M1	DC Motor (-)
6	M1	DC Motor (+)
7	5V	Pressure Sensor Most Bottom Pin when looking at sensor Horizontal and Words are readable and not backwards
8	Analog 0	Pressure Sensor Most Top Pin when looking at sensor Horizontal and Words are readable and not backwards
9	Empty	Empty
10	Empty	Empty
11	Empty	Empty
12	Empty	Empty

Table 7. Pin assignments of 12 pin Connector on Latch Mechanism Control Box to External Components

Figure 8 shows the Schematic of the Latch Mechanism Control Box which was build inside the Latch Mechanism Control Box.

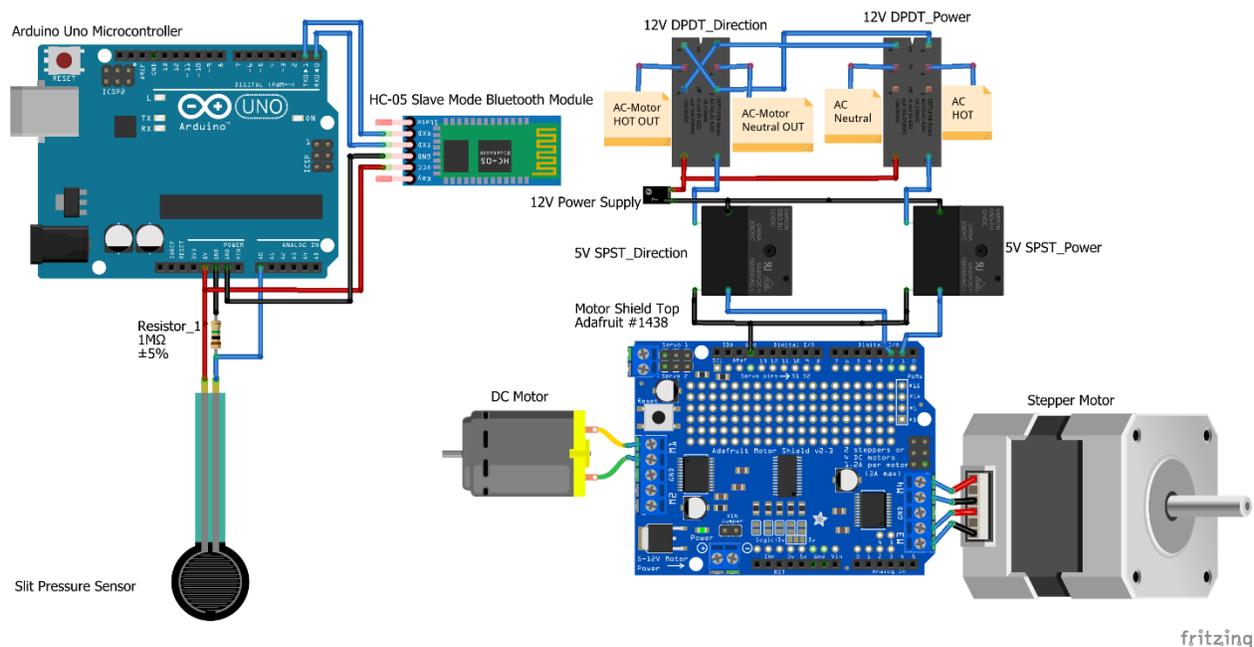


Figure 8. Schematic of Latch Mechanism Control Box

The Code for the Arduino Uno inside the Latch Mechanism Control Box that uses the HC-05 Bluetooth Module in Slave Mode was Built using the following links as references:

- Adafruit Motor Shield software and library download-<https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/install-software>
- Motor Shield Overview- <https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/overview>
- Stepper Motor- <https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/using-stepper-motors>
- DC Motor- <https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/using-dc-motors>
- Pressure Sensor - <https://learn.adafruit.com/force-sensitive-resistor-fsr/using-an-fsr>

The above links are very useful and also have information about libraries that would need to be download in order to properly use the Adafruit Motor Shield V2 with the Arduino Uno.

/*P17310 Latch Arm Control Box

* ** TO LOAD CODE ONTO ARDUINO MAKE SURE BLUETOOTH MODULE IS DISCONNECTED

* This Code is for the Ardunio Inside the Latch Mechanism Control Box which is the Slave mode Bluetooth Module

* The set up for the control box can be found in the following webpage

*

<http://edge.rit.edu/edge/P17310/public/Integrated%20System%20Build%20%26%20Test%20with%20Customer%20Demo>

* This Code will do the following given the following inputs from the Master Mode Arduino through Bluetooth

if given '1' Power Relay Turns On Direction Relay Turns On

if given '2' Power Relay Stays On and Direction Relay Turns Off

if given '3' Both Power and Direction Relay Turn Off, Pressure Sensor Takes a Reading

if given '4' DC Motor CounterClockwise Pulse

if given '5' DC motor Clockwise Pulse

if given '6' Stepper Motor Rotates and lifts the Latch Arm Hook, and brings it back down

*/

```
#include <Wire.h>
```

```
#include <Adafruit_MotorShield.h> //Motor Shield library
```

```
#include "utility/Adafruit_MS_PWMServoDriver.h" //Motor Shield library
```

```
Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x61); //Address to program to the  
Adafruit Motor Shield
```

```
Adafruit_StepperMotor *myStepperMotor = AFMS.getStepper(200, 2); // stepper motor using  
M3 and M4 on the motor shield
```

```
//the stepper motor is 1.8 degrees so one revolution is 200 steps
```

```
Adafruit_DCMotor *myMotor = AFMS.getMotor(1); //Dc motor using M1 on motor shield
```

```
int fsrPin = 0; // the FSR and 1M resistor pulldown are connected to a0 for pressure sensor
```

```
int fsrReading; // the analog reading from the FSR resistor divider
```

```
int ddirection = 12; //direction relay pin digital 12
```

```
int dpower = 13; //power relay pin digital 13
```

```

int state = 0;

void setup() {

    // Setup 2 pins as OUTPUT
    pinMode(ddirection, OUTPUT);
    pinMode(dpower, OUTPUT);

    digitalWrite(ddirection, LOW); //direction for relay system
    digitalWrite(dpower, LOW); //power for relay system

    Serial.begin(38400); // Communication rate of the Bluetooth module has to match the Slave
    Mode communication Rate

    AFMS.begin();
    myStepperMotor->setSpeed(20); // Stepepr Motor 20 rpm
    AFMS.begin(); //frequency set at 1.6 Khz
    myMotor->setSpeed(55); //DC motor 55 rpm (max value is 250 min is 0)
}

void loop() {
    if(Serial.available() > 0){ // Checks whether data is comming from the serial port
        state = Serial.read(); // Reads the data from the serial port
    }

    if ( state =='1') {
        Serial.println("Relay Allowing for Dome Slit to Open");
        digitalWrite(ddirection, HIGH); // Turn direction relay high
        digitalWrite(dpower, HIGH); // turn power relay high
        state = 0;
    }
}

```

```

}

if ( state =='2') {
    Serial.println("Relay Allowing for Dome Slit to Close");
    digitalWrite(dpower, HIGH); // turn Power Relay High
    digitalWrite(ddirection, LOW); // Turn direction Relay Low
    state = 0;
}

if ( state =='3') {
    digitalWrite(ddirection, LOW); // Turn Dirrection Relay Off
    digitalWrite(dpower, LOW); // Turn off Power Relay off
    myMotor->run(RELEASE);

    Serial.println("Pressure Sensor for the Dome Slit Position If Pressure Sensor Anlog Reading
less than 500 It is Open, Otherwise it Closed");
    for (int x = 0; x<1; x++){//scans the for loop once

        fsrReading = analogRead(fsrPin); //recieves abalog reading

        Serial.print("Analog Reading = ");
        Serial.print(fsrReading); // the raw analog reading output

        // We'll have a few thresholds, qualitatively determined
        if (fsrReading > 500) {
            Serial.println(" Dome Slit Closed"); // If preswsure reading is ablove 500 the Slit is Closed

        } else {
            Serial.println(" Dome Slit Opened"); //If preswsure reading is ablove 500 the Slit is open
        }
    }
}

```

```

}

Serial.println(); // print black line

delay(10);}

state = 0;          //

}

if ( state =='4') {

    Serial.println("DC Motor Forward Motion");

    myMotor->run(BACKWARD);//Runs DC motor in reverse for 200ms

    delay (200);

    myMotor->run(FORWARD); //Runs DC motor forward for 25ms

    delay (25);

    myMotor->run(RELEASE);

    state = 0;

} // wait for a second

if ( state =='5') {

    Serial.println("DC Motor Backward Motion");

    myMotor->run(FORWARD); //Runs DC motor for 255ms

    delay (255);

    myMotor->run(BACKWARD);//Runs DC motor in reverse for 15ms

    delay (15);

    myMotor->run(RELEASE); //Power to DC motor is cut off

    state = 0;

} // wait for a second

if ( state =='6') {

    Serial.println("Latch Mechanism Stepper Motor");

```

```

myStepperMotor->step(180, FORWARD, DOUBLE); //Stepper Motor moves forward for 180
steps in double step

myStepperMotor->release(); //power is cut to Stepper so it does not heat up while holding
position

delay(3000); //waaits 3 seconds before proceeding

myStepperMotor->step(180, BACKWARD, DOUBLE); //Stepper Motor moves Reverse for 180
steps in double step

myStepperMotor->release(); //power is cut to Stepper so it does not heat up while holding
position

delay(2000); // waits 2 seconds to finsh function

state = 0; //
}

}

```

The goal of the code is for the Arduino Uno in the Latch Mechanism Control Box to receive Data from the Main Control Box where the DAQ is located via Bluetooth. The data received would result in various actions as seen in table 8. In data in table 8 is of the way the current code is set up where it receives a value from 0 to 6 and each value represents a different output.

Data Value Received Via Bluetooth	Output
0	Power Relay Turns On Direction Relay Turns On, Dome Slit Opens
1	Power Relay Stays On and Direction Relay Turns Off Dome Slit Closes
2	Both Power and Direction Relay Turn Off, Dome Slit Stops Moving. Arduino Receives a Reading from thre Pressure Sensor
3	DC Motor of the Latch Arm Rotates Clockwise for 100ms at 125 RPM
4	DC Motor Counterclockwise Pulse
5	DC motor Clockwise Pulse
6	Stepper Motor Rotates and lifts the Latch Arm Hook, and brings it back down

Table 8. Input and Output of Latch Mechanism Control Box

For the Purpose of testing a circuit with 6 push buttons was built using 1k Resistors, push buttons, the HC-05 Bluetooth Module in Master Mode. The Schematic can be seen in Figure 9. The user sends a

signal by pressing one of the six push buttons and follows the input and outputs of Table 8. Moving From this test set up the DAQ would send a signal to the Arduino rather than a user pressing a push button. The signal is sent from the Master Mode Bluetooth to the Slave Mode Bluetooth inside the Control Box.

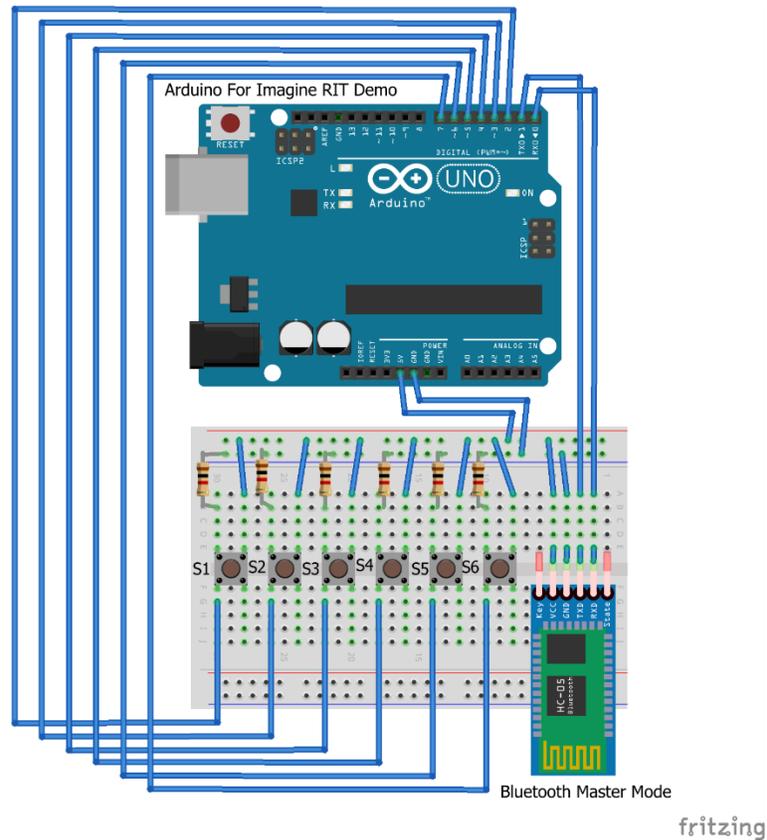


Figure 9. Schematic of Demo and Test Set up of the Master Mode Bluetooth

The following code was downloaded to the Arduino of Figure 9. This code was modified from a Debouncing Code found at one of the tutorials at the Arduino website. [5] The purpose of the code is to take a single reading with a push button is pressed and a value is sent through the Master Mode Bluetooth Module to the Bluetooth inside the Latch Mechanism Control box.

```

/*P17310 Latch Arm Demo For 2017 Imagine RIT
* ***** The following code was modified from the existing code for Debouncing as seen
below
* * * * * TO LOAD CODE ONTO ARDUINO MAKE SURE BLUETOOTH MODULE IS DISCONNECTED
* **This code was used for the Demo at IImagine RIT 2017. Further information on the exhibit
got to

```

*

<http://edge.rit.edu/edge/P17310/public/Customer%20Handoff%20%26%20Final%20Project%20Documentation>

* This Code is for the Arduino and Master Mode Bluetooth Module

*

*

*

Debounce

Each time the input pin goes from LOW to HIGH (e.g. because of a push-button press), the output pin is toggled from LOW to HIGH or HIGH to LOW. There's a minimum delay between toggles to debounce the circuit (i.e. to ignore noise).

by David A. Mellis

modified 30 Aug 2011

by Limor Fried

modified 28 Dec 2012

by Mike Walters

modified 30 Aug 2016

by Arturo Guadalupi

This example code is in the public domain.

<http://www.arduino.cc/en/Tutorial/Debounce>

*/

#define led1 8

```
// constants won't change. They're used here to
// set pin numbers:

const int buttonPin = 2; // the number of the pushbutton pin Digital 2
const int buttonbPin = 3; // the number of the pushbutton pin Digital 3
const int buttoncPin = 4; // the number of the pushbutton pin Digital 4
const int buttoNdPin = 5; // the number of the pushbutton pin Digital 5
const int buttonePin = 6; // the number of the pushbutton pin Digital 6
const int buttonfPin = 7; // the number of the pushbutton pin Digital 7

const int ledPin = 13; // the number of the LED pin

// Variables will change:
int ledState = HIGH; // the current state of the output pin

int state = 20;

int buttonState; // the current reading from the input pin
int buttonbState; // the current reading from the input pin
int buttoncState; // the current reading from the input pin
int buttoNdState; // the current reading from the input pin
int buttoneState; // the current reading from the input pin
int buttonfState; // the current reading from the input pin

int lastButtonState = LOW; // the previous reading from the input pin
int lastButtonbState = LOW; // the previous reading from the input pin
int lastButtoncState = LOW; // the previous reading from the input pin
int lastButtoNdState = LOW; // the previous reading from the input pin
int lastButtoneState = LOW; // the previous reading from the input pin
```

```
int lastButtonfState = LOW; // the previous reading from the input pin
```

```
// the following variables are unsigned long's because the time, measured in milliseconds,  
// will quickly become a bigger number than can be stored in an int.
```

```
unsigned long lastDebounceTime = 0; // the last time the output pin was toggled
```

```
unsigned long debounceDelay = 50; // the debounce time; increase if the output flickers
```

```
unsigned long lastDebouncebTime = 0; // the last time the output pin was toggled
```

```
unsigned long debouncebDelay = 50; // the debounce time; increase if the output flickers
```

```
unsigned long lastDebouncecTime = 0; // the last time the output pin was toggled
```

```
unsigned long debouncecDelay = 50; // the debounce time; increase if the output flickers
```

```
unsigned long lastDebouncedTime = 0; // the last time the output pin was toggled
```

```
unsigned long debouncedDelay = 50; // the debounce time; increase if the output flickers
```

```
unsigned long lastDebounceeTime = 0; // the last time the output pin was toggled
```

```
unsigned long debounceeDelay = 50; // the debounce time; increase if the output flickers
```

```
unsigned long lastDebouncefTime = 0; // the last time the output pin was toggled
```

```
unsigned long debouncefDelay = 50; // the debounce time; increase if the output flickers
```

```
void setup() {
```

```
  pinMode(buttonPin, INPUT);
```

```
  pinMode(buttonbPin, INPUT);
```

```
  pinMode(buttoncPin, INPUT);
```

```
  pinMode(buttondPin, INPUT);
```

```
  pinMode(buttonePin, INPUT);
```

```
  pinMode(buttonfPin, INPUT);
```

```
  pinMode(ledPin, OUTPUT);
```

```
// set initial LED state
digitalWrite(ledPin, ledState);

//digitalWrite(led1, LOW);
Serial.begin(38400); // Default communication rate of the Bluetooth module
}

void loop() {
  if(Serial.available() >0 ){ // Checks whether data is coming from the serial port
    state = Serial.read(); // Reads the data from the serial port
  }

  delay(10);

  // read the state of the switch into a local variable:
  int reading = digitalRead(buttonPin);
  int readingb = digitalRead(buttonbPin);
  int readingc = digitalRead(buttoncPin);
  int readingd = digitalRead(buttondPin);
  int readinge = digitalRead(buttonePin);
  int readingf = digitalRead(buttonfPin);

  // check to see if you just pressed the button
  // (i.e. the input went from LOW to HIGH), and you've waited
  // long enough since the last press to ignore any noise:
```

```

// If the switch changed, due to noise or pressing:
////////////////////////////////////

/**Push Button 1
if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
}

if ((millis() - lastDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer
    // than the debounce delay, so take it as the actual current state:

    // if the button state has changed:
    if (reading != buttonState) {
        buttonState = reading;

        // only toggle the LED if the new button state is HIGH
        if (buttonState == HIGH) {
            Serial.write('1'); // Sends '1' to the master to turn on LED

            ledState = !ledState;
        }
    }
}

////////////////////////////////////

/**Push Button 2
if (readingb != lastButtonbState) {
    // reset the debouncing timer
    lastDebouncebTime = millis();
}

```

```

}

if ((millis() - lastDebouncebTime) > debouncebDelay) {
    // whatever the reading is at, it's been there for longer
    // than the debounce delay, so take it as the actual current state:

    // if the button state has changed:
    if (readingb != buttonbState) {
        buttonbState = readingb;

        // only toggle the LED if the new button state is HIGH
        if (buttonbState == HIGH) {
            Serial.write('2'); // Sends '1' to the master to turn on LED

            ledState = !ledState;
        }
    }
}

////////////////////////////////////
/**Push Button 3
if (readingc != lastButtoncState) {
    // reset the debouncing timer
    lastDebouncecTime = millis();
}

if ((millis() - lastDebouncecTime) > debouncecDelay) {
    // whatever the reading is at, it's been there for longer
    // than the debounce delay, so take it as the actual current state:

```

```

// if the button state has changed:
if (readingc != buttoncState) {
  buttoncState = readingc;

  // only toggle the LED if the new button state is HIGH
  if (buttoncState == HIGH) {
    Serial.write('3'); // Sends '1' to the master to turn on LED

    ledState = !ledState;
  }
}

////////////////////////////////////
/**Push Button 4

// If the switch changed, due to noise or pressing:
if (readingd != lastButtondState) {
  // reset the debouncing timer
  lastDebounceTime = millis();
}

if ((millis() - lastDebounceTime) > debounceDelay) {
  // whatever the reading is at, it's been there for longer
  // than the debounce delay, so take it as the actual current state:

  // if the button state has changed:
  if (readingd != buttondState) {
    buttondState = readingd;

```

```

// only toggle the LED if the new button state is HIGH
if (buttonState == HIGH) {
Serial.write('4'); // Sends '1' to the master to turn on LED

    ledState = !ledState;
}
}
}
////////////////////////////////////
/**Push Button 5
if (reading != lastButtonState) {
// reset the debouncing timer
lastDebounceTime = millis();
}

if ((millis() - lastDebounceTime) > debounceDelay) {
// whatever the reading is at, it's been there for longer
// than the debounce delay, so take it as the actual current state:

// if the button state has changed:
if (reading != buttonState) {
    buttonState = reading;

// only toggle the LED if the new button state is HIGH
if (buttonState == HIGH) {
    Serial.write('5'); // Sends '1' to the master to turn on LED

    ledState = !ledState;
}
}
}

```

```

}
}
////////////////////////////////////
/**Push Button 6
if (readingf != lastButtonfState) {
    // reset the debouncing timer
    lastDebouncefTime = millis();
}

if ((millis() - lastDebouncefTime) > debouncefDelay) {
    // whatever the reading is at, it's been there for longer
    // than the debounce delay, so take it as the actual current state:

    // if the button state has changed:
    if (readingf != buttonfState) {
        buttonfState = readingf;

        // only toggle the LED if the new button state is HIGH
        if (buttonfState == HIGH) {
            Serial.write('6'); // Sends '1' to the master to turn on LED

            ledState = !ledState;
        }
    }
}

////////////////////////////////////

// set the LED:
digitalWrite(ledPin, ledState);

```

```

// save the reading. Next time through the loop,

// it'll be the lastButtonState:

lastButtonState = reading;

lastButtonbState = readingb;

lastButtoncState = readingc;

lastButtondState = readingd;

lastButtoneState = readinge;

lastButtonfState = readingf;

}

```

Connecting thr computer to the Arduino inside the Latch Mechanism Control Box the following figure 10. Shows the outputs of the commands of pressing the push buttons in the order of: 1, 3, 2, 3, 4, 5, 6.

*******If 6 is pressed the code will save pressed buttons but not act on them until Stepper motor is done moving. If buttons are pressed while stepper motor is moving, when it is done moving Arduino will have to be reset if the user started pressing buttons randomly to see if anything works because the Arduino will follow the commands and may result in unwanted outputs.**

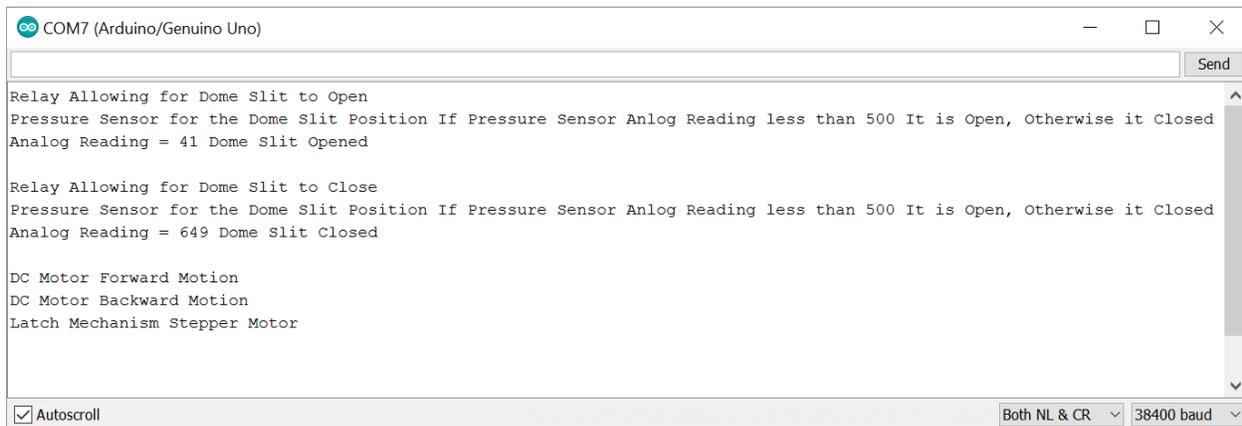


Figure 10. Screenshot of the Latch Mechanism Control Box output Serial Monitor

Trouble Tracking

Problem Number	Identifying & Selecting Problem PSP 1	Analyzing Problem PSP 2	Generating Potential Solutions PSP 3	Selecting & Planning Solution PSP 4	Implementing Solution PSP 5	Evaluating Solution PSP 6
	R1	R2	R3	Y4	Y5	G6
1	DC Motor for Latch Mechanism is drawing too much current	Due to the high torque needed to rotate the latch arm the DC motor is drawing current over 1.5amps which goes over the limit of the Arduino Motor Shield	Finding a New 12V Motor/Controller or Finding a New 24 V Motor/Controller as long it downs use high current because there is a 6amp limit from the power rails			
2	Latch Arm too heavy causing high current draw of DC Motor	The weight of the aluminium latch arm causes DC motor to draw high current due to higher torque needed to move	3D printing the Latch Arm would lighten the load of the DC motor or drilling holes through Aluminium to lighten the weight	A 3D version of the Latch Arm was printed	The 3D printed Latch Arm was connected to the DC Motor for Testing	3D printed Latch arm was tried which was lighter and DC motor did not draw as much current as the aluminum but it is still high. The problem still remains
3	When the DC Motor rotates the Latch Arm, the wires of the Stepper Motor may get tangled	Due to the Stepper Motor being mounted on a moving part that has the potential to rotate over 360 degrees, the wires of the stepper motor have a high chance of getting tangled	Configure DC motor so it does not rotate over 360 degrees from the initial Starting Point and leave some wire slack to Arm can rotate without the wires of the Stepper Motor getting tangled			

Table 9. TRouble Tracking For Latch Mechanism Control Box

Troubleshooting The Test Set-Up

Problem	Solution
Motors Do Not Turn on	<ul style="list-style-type: none"> Make sure there is a 12V power source connected Make sure set up was properly done by referencing https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/overview
Motor turns on but do not rotate	<ul style="list-style-type: none"> Too much torque may be applied and motor cant handle it Lower torque applied to it
Theres no power inside the Latch Mechanism Control box	<ul style="list-style-type: none"> Make sure the toggle switch in front of the box is set to automatic mode. Remove power and safely check wiring
HC-05 Bluetooth Module getting HOT	<ul style="list-style-type: none"> Immediately but Safely remove power from system Wait until module has cooled down because now Module is a Burning Hazard After it has cooled Check wiring as seen in figure 6 If module not turn On it will need to be replaced
Relays flipping randomly	<ul style="list-style-type: none"> Immediately but Safely remove power from system

	<ul style="list-style-type: none"> • The relays will flip randomly when code is downloaded • Make sure relays not loose from their mounts
Code not downloading to Arduino	<ul style="list-style-type: none"> • Make sure Bluetooth Module not connected

Future Goals

In the future first the Motors will have to worked on due to the problem of the DC motor pulling too much current and the Adafruit Motor shield works best up to 2Amps preferably and current the DC motor is pulling above 3Amps which will cause damage to the motor shield and Arduino. Getting a higher current rated motor shield is not an ideal solution because there is also a current limit that the Power Rails can handle before a fuse blowing. Our solution was 3-D printed Arm instead of the Aluminum arm which drew less current but it was still over what was desired. Keep in Mind that the Arm still has to be able to pull the chain to unlatch the slit so the Arm can't be too light otherwise it make bend or break trying to unlatch.

Instead of using push buttons to actuate the components inside the Latch Mechanism Control Box, the Labview Program would send data to the DAQ at the Main Control Box which would send commands to an Arduino and through Bluetooth communication it would send the commands to the Bluetooth module inside the Control to actuate the components.

Lastly there is still potential on moving the Latch Mechanism forward with possible reconsiderations on parts used and code integration with the Main System with safety always being a priority when working on improvements.

References

[1] [https://www.itead.cc/wiki/Serial_Port_Bluetooth_Module_\(Master/Slave\)_:_HC-05](https://www.itead.cc/wiki/Serial_Port_Bluetooth_Module_(Master/Slave)_:_HC-05)

[2] <http://history-computer.com/ModernComputer/Basis/modem.html>

[3] https://www.digikey.com/product-detail/en/PPTC061LFBN-RC/S7004-ND/810145?WT.mc_id=IQ_7595_G_pla810145&wt.srch=1&wt.medium=cpc&&gclid=CjwKEAjw3KDIBRCzOKvZIJ7k4TgSJABDqOK7Blk8l8eaiJlCwAxlmhDUCGraPAY8rJzp0RnMZBR72hoChh7w_wcB

[4] <http://www.martyncurrey.com/connecting-2-arduinis-by-bluetooth-using-a-hc-05-and-a-hc-06-pair-bind-and-link/>

[5] <http://www.arduino.cc/en/Tutorial/Debounce>

Mentioned Links

Motor Shield Overview- <https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/overview>

Stepper Motor- <https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/using-stepper-motors>

DC Motor- <https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/using-dc-motors>

Pressure Sensor - <https://learn.adafruit.com/force-sensitive-resistor-fsr/using-an-fsr>

**Wiring Schematics was built using Open Source Software fritzing-<http://fritzing.org/home/>