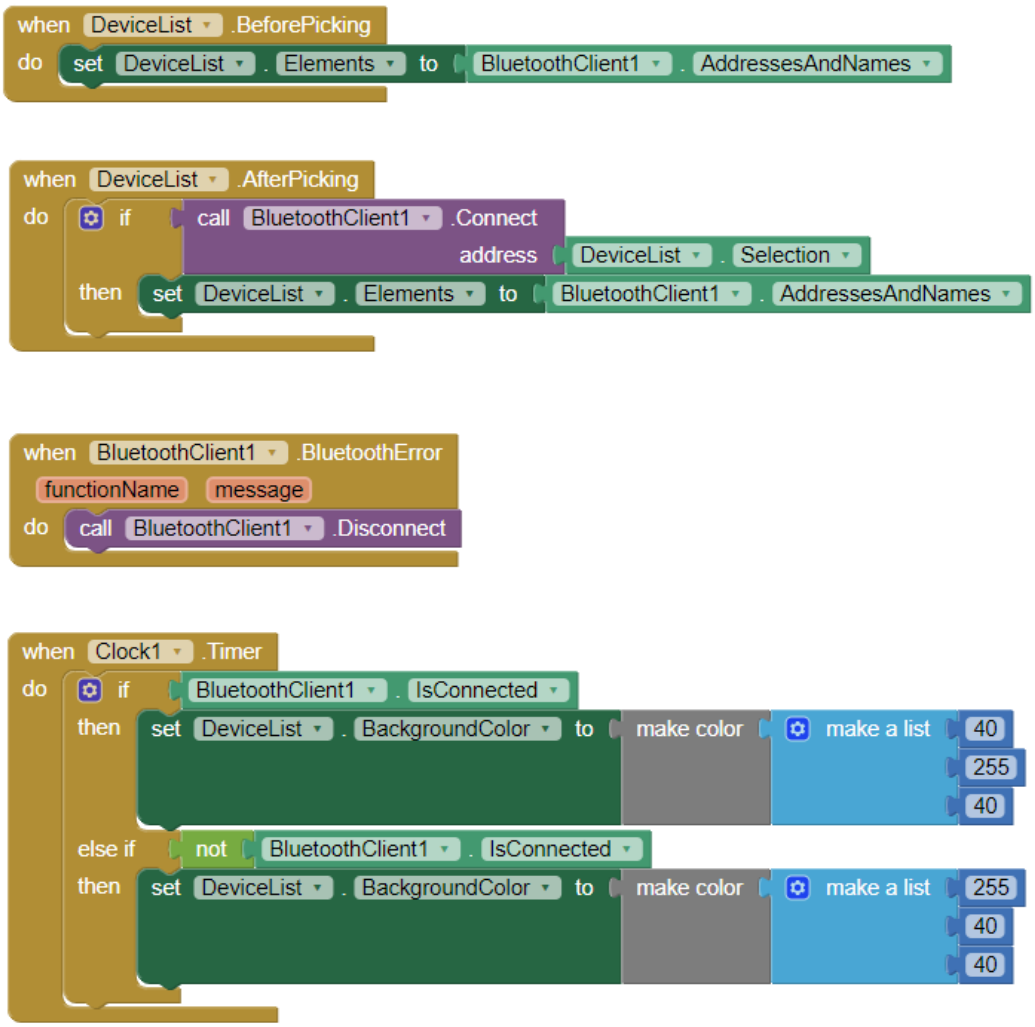
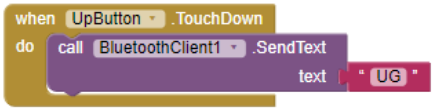
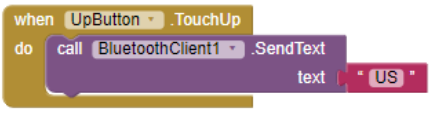
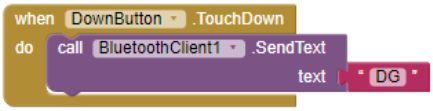
App Design:

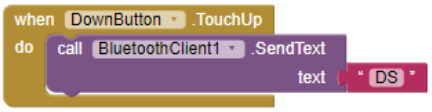
App Code:

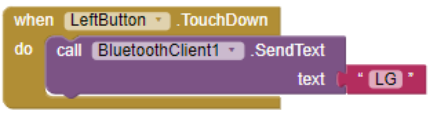


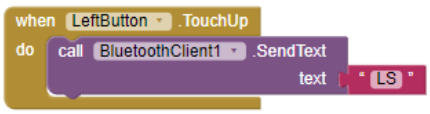

when UpButton .TouchDown
do call BluetoothClient1 .SendText
text " UG "

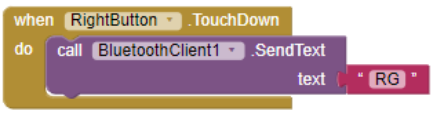

when UpButton .TouchUp
do call BluetoothClient1 .SendText
text " US "

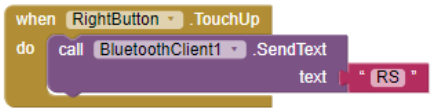

when DownButton .TouchDown
do call BluetoothClient1 .SendText
text " DG "

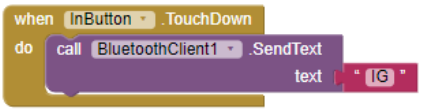

when DownButton .TouchUp
do call BluetoothClient1 .SendText
text " DS "

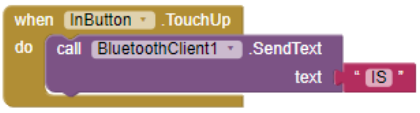

when LeftButton .TouchDown
do call BluetoothClient1 .SendText
text " LG "

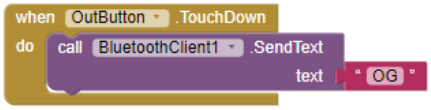

when LeftButton .TouchUp
do call BluetoothClient1 .SendText
text " LS "

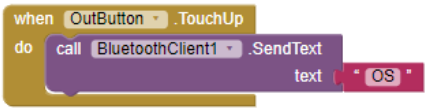

when RightButton .TouchDown
do call BluetoothClient1 .SendText
text " RG "

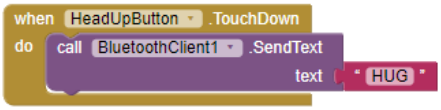

when RightButton .TouchUp
do call BluetoothClient1 .SendText
text " RS "

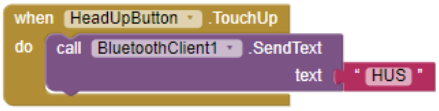

when InButton .TouchDown
do call BluetoothClient1 .SendText
text " IG "

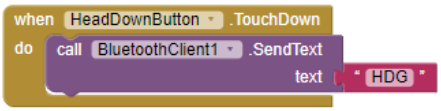

when InButton .TouchUp
do call BluetoothClient1 .SendText
text " IS "

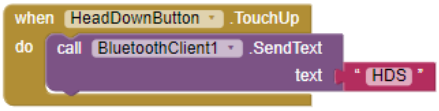

when OutButton .TouchDown
do call BluetoothClient1 .SendText
text " OG "

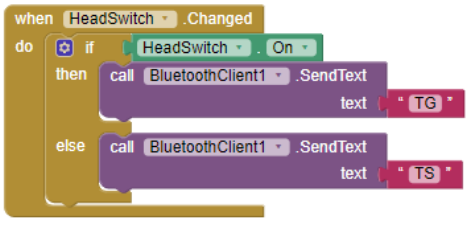

when OutButton .TouchUp
do call BluetoothClient1 .SendText
text " OS "

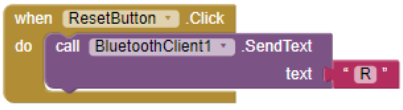

when HeadUpButton .TouchDown
do call BluetoothClient1 .SendText
text " HUG "

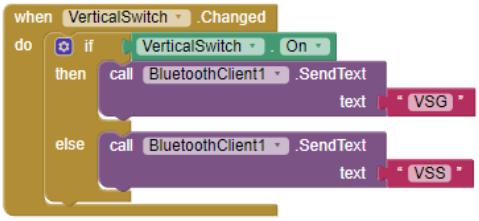

when HeadUpButton .TouchUp
do call BluetoothClient1 .SendText
text " HUS "

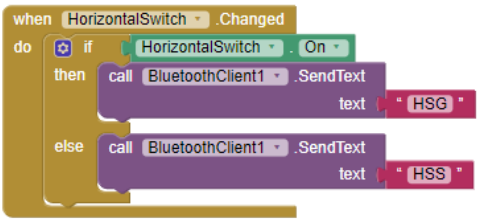

when HeadDownButton .TouchDown
do call BluetoothClient1 .SendText
text " HDG "


when HeadDownButton .TouchUp
do call BluetoothClient1 .SendText
text " HDS "


when HeadSwitch .Changed
do if HeadSwitch . On
then call BluetoothClient1 .SendText
text " TG "
else call BluetoothClient1 .SendText
text " TS "


when ResetButton .Click
do call BluetoothClient1 .SendText
text " R "


when VerticalSwitch .Changed
do if VerticalSwitch . On
then call BluetoothClient1 .SendText
text " VSG "
else call BluetoothClient1 .SendText
text " VSS "


when HorizontalSwitch .Changed
do if HorizontalSwitch . On
then call BluetoothClient1 .SendText
text " HSG "
else call BluetoothClient1 .SendText
text " HSS "

Arduino Code:

```
//including necessary libraries
#include <SoftwareSerial.h>
#include <Servo.h>
#include <Stepper.h>

//Creating necessary objects and variables for the bluetooth module
SoftwareSerial Bluetooth(2, 3);
String dataIn = "";

//Initialising servos as instances of the Servo class and defining variables for their angles
Servo base;
Servo shoulder;
Servo elbow;
Servo wrist;
float baseAngle;
float shoulderAngle;
float elbowAngle;
float wristAngle;

//Creating/defining variables for the stepper motor and creating an instance of the Stepper class for it
const float STEPS_PER_REV = 32;
const float GEAR_RED = 64;
const float STEPS_PER_OUT_REV = STEPS_PER_REV * GEAR_RED;
const int headSpeed = 1000;
int stepsRequired;
Stepper head(STEPS_PER_REV, 8, 10, 9, 11);

//Creating the boolean values for indication of toggled modes and actions done, used in logic paths later in the code
bool twist = false;
bool scratchVertical = false;
bool scratchHorizontal = false;
bool hasScratched;

//Creating values to store the change in angle when scratching
float verticalOffset = 0;
float horizontalOffset = 0;

//Creating values to store the direction of the scratch as it happens
bool verticalIncrease = true;
bool horizontalIncrease = true;

//Creating variables to determine the factors of the scratch
float scratchPeriod = 10;
float verticalAmplitude = 15;
float horizontalAmplitude = 20;

//The set up function
void setup() {
  //Setting up the bluetooth connection and timeout period
  Bluetooth.begin(9600);
  Bluetooth.setTimeout(1);

  //A loop to keep the motors limp until data is available
  while (Bluetooth.available() == 0) {
    delay(10);
  }

  //Connecting the servos to their corresponding pins
  base.attach(4);
  shoulder.attach(5);
  elbow.attach(6);
  wrist.attach(7);

  //Calling the reset functino to direct the servos to their starting positions
  reset();
}
```

```
//The loop function to be repeated throughout the coded
void loop() {
  hasScratched = false;

  //Checking if there is incoming bluetooth data
  if (Bluetooth.available() > 0) {
    //Assigning the bluetooth data to a variable
    dataIn = Bluetooth.readString();

    //Checking if the data is for the head's scratching to be turned on and making subsequent adjustments
    if (dataIn == "TG"){
      twist = true;
    }
    //Checking if the data is for the head's scratching to be turned off and making subsequent adjustments
    else if (dataIn == "TS"){
      twist = false;
    }
    //Checking if the data is for the vertical scratching to be turned on and making subsequent adjustments
    else if (dataIn == "VSG"){
      scratchVertical = true;
      verticalOffset = 0;
    }
    //Checking if the data is for the vertical scratching to be turned off and making subsequent adjustments
    else if (dataIn == "VSS"){
      scratchVertical = false;
      shoulder.write(shoulderAngle);
      elbow.write(elbowAngle);
      wrist.write(wristAngle);
    }
    //Checking if the data is for the horizontal scratching to be turned on and making subsequent adjustments
    else if (dataIn == "HSG"){
      scratchHorizontal = true;
      horizontalOffset = 0;
    }
    //Checking if the data is for the horizontal scratching to be turned off and making subsequent adjustments
    else if (dataIn == "HSS"){
      scratchHorizontal = false;
      base.write(baseAngle);
    }
  }

              //Checking if the most recent data is for the "Up Go" instruction
              if (dataIn == "UG") {
                //Checking if the elbow is still within the custom limit
                if (elbowAngle > 26 && shoulderAngle > 90) {
                  //Using the rotate function to adjust the multiple necessary motors
                  rotate(elbow, elbowAngle, -2);
                  rotate(shoulder, shoulderAngle, -1);
                  rotate(wrist, wristAngle, -1);
                }
              }

              //Checking if the most recent data is for the "Down Go" instruction
              else if (dataIn == "DG") {
                //Testing if the extension of the arm is within the custom limit
                if (elbowAngle < 140) {
                  //Using the rotate function to adjust the multiple necessary motors
                  rotate(elbow, elbowAngle, 2);
                  rotate(shoulder, shoulderAngle, 1);
                  rotate(wrist, wristAngle, 1);
                }
              }

              //Checking if the most recent data is for the "Left Go" instruction
              else if (dataIn == "LG") {
                //Using the rotate function to rotate the base servo left by 1.5 degrees
                rotate(base, baseAngle, -1.5);
              }

              //Checking if the most recent data is for the "Right Go" instruction
              else if (dataIn == "RG") {
                //Using the rotate function to rotate the base servo right by 1.5 degrees
                rotate(base, baseAngle, 1.5);
              }

              //Checking if the most recent data is for the "In Go" instruction
              else if (dataIn == "IG") {
                //Using the rotate function to rotate the shoulder servo down by 1 degree
                rotate(shoulder, shoulderAngle, 1);
              }

              //Checking if the most recent data is for the "Out Go" instruction
              else if (dataIn == "OG") {
                //Checking if the joint is within the custom limit of 90 degrees
```

```
  //Checking if the joint is within the custom limit of 90 degrees
  if (shoulderAngle > 90) {
    //Using the rotate function to rotate the shoulder servo up by 1 degree
    rotate(shoulder, shoulderAngle, -1);
  }
}

//Checking if the most recent data is for the "Head Up Go" instruction
else if (dataIn == "HUG") {
  //Using the rotate function to rotate the wrist servo up by 2 degrees
  rotate(wrist, wristAngle, 2);
}

//Checking if the most recent data is for the "Head Down Go" instruction
else if (dataIn == "HDG") {
  //Using the rotate function to rotate the wrist servo down by 2 degrees
  rotate(wrist, wristAngle, -2);
}

//Checking if the most recent data is for the "Reset Position" instruction
else if (dataIn == "R") {
  //Calling the reset function to reset the servos and variables, and reseting the incoming data as to not run this path twice
  reset();
  dataIn = "";
}

//Checking if the twist value is true
if (twist){
  //Moving the head spikes by a small turn, roughly equal to a 50 millisecond delay
  head.setSpeed(headSpeed);
  stepsRequired = STEPS_PER_OUT_REV / 75;
  head.step(stepsRequired);
  hasScratched = true;
}

//Checking if the vertical scratch value is true
if (scratchVertical){
  //Checking if the up section of the scratch motion is in effect
  if (verticalIncrease) {
    //Adjusting variables and moving motors to the new location
    verticalOffset -= (verticalAmplitude / scratchPeriod);
    shoulder.write(shoulderAngle + verticalOffset);
    elbow.write(elbowAngle + verticalOffset * 2);
    wrist.write(wristAngle + verticalOffset);
    //Ending the up movement of the scratch motion if the upper limit has been reached
    if (verticalOffset <= -verticalAmplitude){
      verticalIncrease = false;

        }
      }
      //Actions when in the downwards phase of scratching
      else {
        //Adjusting variables and moving motors to the new location
        verticalOffset += (verticalAmplitude / scratchPeriod);
        shoulder.write(shoulderAngle + verticalOffset);
        elbow.write(elbowAngle + verticalOffset * 2);
        wrist.write(wristAngle + verticalOffset);
        //Ending the down movement of the scratch motion if the lower limit has been reached
        if (verticalOffset >= verticalAmplitude){
          verticalIncrease = true;
        }
      }
    }

    //Checking if the horizontal scratch value is true
    if (scratchHorizontal){
      //Checking if the scratch motion is currently in its right moving phase
      if (horizontalIncrease){
        //Adjusting variables and moving motors
        horizontalOffset += (horizontalAmplitude / scratchPeriod);
        base.write(baseAngle + horizontalOffset);
        //Ending the right movement of the scratch motion if the upper limit has been reached
        if (horizontalOffset >= horizontalAmplitude){
          horizontalIncrease = false;
        }
      }
      //Actions when in the left phase of scratching
      else {
        //Adjusting variables and moving motors
        horizontalOffset -= (horizontalAmplitude / scratchPeriod);
        base.write(baseAngle + horizontalOffset);
        //Ending the left movement of the scratch motion if the lower limit has been reached
        if (horizontalOffset <= -horizontalAmplitude){
          horizontalIncrease = true;
        }
      }
    }

    //Delaying the next loop as to space apart the actions, assuming the twist delay is not already used
    if (not hasScratched) {
      delay(50);
    }
  }
```

```
//Defining the reset function
void reset() {
  //Setting the angle variables for each motor back to the initial position
  baseAngle = 90;
  shoulderAngle = 130;
  elbowAngle = 90;
  wristAngle = 110;

  //Moving the motors to the new angles
  base.write(baseAngle);
  shoulder.write(shoulderAngle);
  elbow.write(elbowAngle);
  wrist.write(wristAngle);
}



//Defining the rotate function, taking the servo, its angle variable and the amount to increment as parameters
void rotate(Servo &thisServo, float &angle, float increment) {
  //Checking if the increment is negative
  if (increment < 0) {
    //Checking if the angle is at its minimum
    if (angle > 0) {
      //decreasing the angle variable by the increment and moving the servo to this new angle
      angle += increment;
      thisServo.write(angle);
    }
  }
  //Conditional statement of the increment being positive
  else {
    //Checking if the angle is at its maximum
    if (angle < 180) {
      //increasing the angle variable by the increment and moving the servo to this new angle
      angle += increment;
      thisServo.write(angle);
    }
  }
}
```