H Code Snippets

H.1 Writing Motor Angles to Arduino

```
// Sends the appropriate signals to the Arduino to set the motor angles
public void setMotorAngles()
{
    // Already eliminated extreme angles in the Experiment class and no NaN values
        should have been added
    byte[] instructionBuffer = new byte[2];
    int oldAngle, newAngle, toWrite;
    while (anglesNotEqual())
    {
        for (int i = 0; i < 4; i++)
        {
            instructionBuffer[0] = Convert.ToByte(i + 1);
            oldAngle = currentMotorAngles[i];
            newAngle = targetMotorAngles[i];

            if (oldAngle < newAngle) oldAngle++;
            else if (oldAngle > newAngle) oldAngle--;
            else continue;

            toWrite = oldAngle;

            if (usingOffset && i == 0) toWrite += m1Offset;
            if (usingOffset && i == 2) toWrite += m3Offset;

            instructionBuffer[1] = Convert.ToByte(toWrite);

            lock(arduinoLock) // So other threads don't simultaneously write to
                Arduino and cause incorrect serial transmission
            {
                currentPort.Write(instructionBuffer, 0, 2);
            }

            Thread.Sleep(setMotorDelay);
            updateCurrentMotorAngles(i, oldAngle);
        }
    }
}
```

H.2 Receiving Instructions on the Arduino

```
void loop() {
...
  if (sentFromPC()) {

    int controlCode = Serial.read();
    int instruction = Serial.read();
    if(controlCode > 0 && controlCode < 5) {
      motorInstruction(controlCode, instruction);
    }
    else if(controlCode == 7) {
      triggerInstruction(instruction);
    }
  }

}

bool sentFromPC() {
  if(connectedToPC && Serial.available() == 2) {
    return true;
  }
  else {
    return false;
  }
}

void motorInstruction(int selectedServo, int angle) {
  switch(selectedServo) {
      case 1:
        currentServo = myservo1;
        break;
      case 2:
        currentServo = myservo2;
        break;
      case 3:
        currentServo = myservo3;
        break;
      case 4:
        currentServo = myservo4;
        break;
      default:
        break;
    }
    currentServo.write(angle);
}

void triggerInstruction(int instruction) {

    if(instruction == 7) {
      respondTriggerValue();
    }
    else if(instruction == 1) {
      digitalWrite(magnetControlPin, HIGH);
    }
    else if(instruction == 0) {
      digitalWrite(magnetControlPin, LOW);
    }
}
```

H.3 Regression Function

```
private double[] NWRegression(float[] inputVectorTarget, RegressionInput inputType,
    float alpha = startingAlpha)
{
    int numOutputDimensions = getNumOutputDimensions(inputType);
    double[] outputVector = new double[numOutputDimensions];
    double[] sumNumerator = new double[numOutputDimensions];
    double sumDenominator = 0;
    float[] newInputVector = getCopyVector(inputVectorTarget);

    if (inputType == RegressionInput.MOTORS)
    {
        convertMotors(newInputVector, true);
    }

    // loop through entire set of data
    for (int i = 0; i < storedCalibrationData.Count; i++)
    {
        float kernelInput = getKernelInput(i, inputType, alpha, newInputVector);
        float[] currentYValue = getcurrentYValue(i, inputType);
        for (int k = 0; k < currentYValue.Length; k++)
        {
            sumNumerator[k] += currentYValue[k] * kernelFunction(kernelInput);
        }
        sumDenominator += kernelFunction(kernelInput);
    }


    for (int k = 0; k < numOutputDimensions; k++)
    {
        outputVector[k] = sumNumerator[k] / sumDenominator;
    }

    if (inputType != RegressionInput.MOTORS)
    {
        convertOutputMotors(outputVector, false);
    }

    return outputVector;
}
```

H.4 Logic and Angle Setting in Different Threads for User Study

```
// Sets the user study going, starts the thread to constantly update the motor
    angles
public void startStudy(UserStudyType type)
{
    activeStudy = new UserStudy(type, randomiseGesturing);
    experimentLive = true;
    new Task(liveExperimentThreadLoop).Start();
    runUserStudy();
    experimentLive = false;
    controller.zeroMotors();
    if(type == UserStudyType.GESTURING)
    {
        activeStudy.printRandomisedOrder();
    }
}

// Repeatedly sets the motor angles (delay is present within setMotorAngles
    function)
private void liveExperimentThreadLoop()
{
    while (experimentLive)
    {
        setMotorAngles();
    }
}

// Runs user study by informing object of base position and trigger, and fetching
    appropriate position
private void runUserStudy()
{
    bool triggerPress;
    float[] relativeTargetPosition;
    if (!activeStudy.isInitialised())
    {
        // Check that it has been initialise before attempting to run
        Console.WriteLine("unable to initialise user study");
        return;
    }
    relativeTargetPosition = activeStudy.getRelativeTargetPosition();
    runningStudy = true;
    while(runningStudy)
    {
        triggerPress = getTrigger();
        runningStudy = activeStudy.update(basePosition, triggerPress);

        if(triggerPress)
        {
            zeroMotorAngles();
            controller.activateMagnet(true);
        }
        else if (runningStudy)
        {
            controller.activateMagnet(false);
            getMotorAnglesForTargetPoint(relativeTargetPosition);
        }
        Thread.Sleep(newTargetDelay);
    }
}
```