```
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_MS_PWMServoDriver.h"

// Pair motors to physical pins
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *motorL = AFMS.getMotor(1);
Adafruit_DCMotor *motorR = AFMS.getMotor(2);

// Pair sensors to physical pins
const int sensorPinLL = A3;
const int sensorPinCL = A2;
const int sensorPinCR = A1;
const int sensorPinRR = A0;

// Reflectivity cutoff value
const int calibrationValue = 40;

// Initialize request value to 0 (stopped)
int request = 0;

// Create time variables
unsigned long startTime, elapsedTime;

// Create PID variables
int P, I, D;
int lastError = 0;
float Kp = .3;
float Ki = 0;
float Kd = .5;

// Set max and base motor speeds
uint8_t maxspeedraw = 50;
uint8_t basespeedraw = 30;
uint8_t maxspeed, basespeed;

// Initialize output string to empty
String output = "";

void setup()
{
  // Initialize motor controller
  AFMS.begin();

  // start the serial port
  long baudRate = 9600;
  Serial.begin(baudRate);
  Serial.setTimeout(1);

  // Initialize all sensors
  pinMode(sensorPinLL, INPUT);
  pinMode(sensorPinCL, INPUT);
  pinMode(sensorPinCR, INPUT);
  pinMode(sensorPinRR, INPUT);

  // Ensure motors are released
  motorL->run(RELEASE);
  motorR->run(RELEASE);

  // Only continue while serial connection is available
  // This means nothing happens until the Python program is run
  while (!Serial.available())
  {
  }

  // Wait to hear a non-zero value from Python before starting
  while (request == 0)
  {
    request = Serial.readString().toInt();
  }

  // Get start time for time tracking
  startTime = millis();
}

void loop()
{
  if (request != 0)
  {
    // Request can be 1 or 2
    // Creates 2 speed options
    maxspeed = maxspeedraw * request;
    basespeed = basespeedraw * request;

    PID_control();
  }
  else
  {
    motorL->run(RELEASE);
    motorR->run(RELEASE);
    exit;
  }
}

/**
 * Using aggregate sensor value, control motors to follow line
 */
void PID_control()
{
  uint16_t position = sensorValue();

  // 500 is "center", so this centers the sensor value
  int error = 500 - position;

  // Calculate PID response
  P = error;
  I = I + error;
  D = error - lastError;
  lastError = error;

  int motorspeed = P * Kp + I * Ki + D * Kd;

  int motorspeedL = basespeed + motorspeed;
  int motorspeedR = basespeed - motorspeed;

  // Ensure values don't exceed range (0, maxspeed)
  if (motorspeedL > maxspeed)
  {
    motorspeedL = maxspeed;
  }
  if (motorspeedR > maxspeed)
  {
    motorspeedR = maxspeed;
  }
  if (motorspeedL < 0)
  {
    motorspeedL = 0;
  }
  if (motorspeedR < 0)
  {
    motorspeedR = 0;
  }
  // Set speeds of motors
  motorL->setSpeed(motorspeedL);
  motorR->setSpeed(motorspeedR);
  motorL->run(FORWARD);
  motorR->run(FORWARD);

  // Print full string of values to serial port
  output = output + motorspeedL + "," + motorspeedR;
  Serial.println(output);
}

/**
 * Read sensors and calculate an aggregate reading
 */
uint16_t sensorValue()
{
  // Read all 4 sensor values
  float rawLL = analogRead(sensorPinLL);
  float rawCL = analogRead(sensorPinCL);
  float rawCR = analogRead(sensorPinCR);
  float rawRR = analogRead(sensorPinRR);

  // Determine which sensors are on the line
  bool sensorLL = onLine(rawLL);
  bool sensorCL = onLine(rawCL);
  bool sensorCR = onLine(rawCR);
  bool sensorRR = onLine(rawRR);

  // Add elapsed time and raw sensor values to output string
  output = String(float(millis() - startTime))
            + "," + rawLL + "," + rawCL
            + "," + rawCR + "," + rawRR + ",";

  /**
   * Determine aggregate sensor value based on
   * combination of sensor values.
   */
  if (sensorLL && !sensorCL && !sensorCR && !sensorRR)
  {
    return 800;
  }
  else if (sensorLL && sensorCL && !sensorCR && !sensorRR)
  {
    return 700;
  }
  else if (!sensorLL && sensorCL && !sensorCR && !sensorRR)
  {
    return 600;
  }
  else if (!sensorLL && sensorCL && sensorCR && !sensorRR)
  {
    return 500;
  }
  else if (!sensorLL && !sensorCL && sensorCR && !sensorRR)
  {
    return 400;
  }
  else if (!sensorLL && !sensorCL && sensorCR && sensorRR)
  {
    return 300;
  }
  else if (!sensorLL && !sensorCL && !sensorCR && sensorRR)
  {
    return 200;
  }
```

```
}

/**
 * Determine whether a given sensor is on the line.
 */
bool onLine(float sensorRaw)
{
  if (sensorRaw > calibrationValue)
  {
    return true;
  }
  else
  {
    return false;
  }
}

```