**ARDUINO CODE COMBINING FUZZY LOGIC SYSTEM AND ANFIS SYSTEM**

//Fuzzy Logic

#define FIS\_TYPE float

#define FIS\_RESOLUSION 101

#define FIS\_MIN -3.4028235E+38

#define FIS\_MAX 3.4028235E+38

typedef FIS\_TYPE(\*\_FIS\_MF)(FIS\_TYPE, FIS\_TYPE\*);

typedef FIS\_TYPE(\*\_FIS\_ARR\_OP)(FIS\_TYPE, FIS\_TYPE);

typedef FIS\_TYPE(\*\_FIS\_ARR)(FIS\_TYPE\*, int, \_FIS\_ARR\_OP);

//ANFIS

#define FIS1\_TYPE float

#define FIS1\_MIN -3.4028235E+38

#define FIS1\_MAX 3.4028235E+38

typedef FIS1\_TYPE(\*\_FIS1\_MF)(FIS1\_TYPE, FIS1\_TYPE\*);

typedef FIS1\_TYPE(\*\_FIS1\_ARR\_OP)(FIS1\_TYPE, FIS1\_TYPE);

typedef FIS1\_TYPE(\*\_FIS1\_ARR)(FIS1\_TYPE\*, int, \_FIS1\_ARR\_OP);

//Fuzzy Logic

const int fis\_gcI = 4;

const int fis\_gcO = 1;

const int fis\_gcR = 81;

//ANFIS

const int fis1\_gcI = 2;

const int fis1\_gcO = 1;

const int fis1\_gcR = 25;

//Input & Output of Fuzzy Logic

FIS\_TYPE g\_fisInput[fis\_gcI];

FIS\_TYPE g\_fisOutput[fis\_gcO];

//Input & Output of ANFIS

FIS1\_TYPE g\_fis1Input[fis1\_gcI];

FIS1\_TYPE g\_fis1Output[fis1\_gcO];

FIS1\_TYPE g\_fis2Input[fis1\_gcI];

FIS1\_TYPE g\_fis2Output[fis1\_gcO];

FIS1\_TYPE g\_fis3Input[fis1\_gcI];

FIS1\_TYPE g\_fis3Output[fis1\_gcO];

FIS1\_TYPE g\_fis4Input[fis1\_gcI];

FIS1\_TYPE g\_fis4Output[fis1\_gcO];

//Initialize IR sensors

int irLA\_1=22;

int irLA\_2=23;

int irLB\_1=24;

int irLB\_2=25;

int irLC\_1=26;

int irLC\_2=27;

int irLD\_1=28;

int irLD\_2=29;

//initialize LEDs

int rA = 42;

int yA = 43;

int gA = 44;

int rB = 45;

int yB = 46;

int gB = 47;

int rC = 48;

int yC = 49;

int gC = 50;

int rD = 51;

int yD = 52;

int gD = 53;

///////////////////LANE 1

int IRinLane1 = 22;

int IRoutLane1 = 23;

int addcurrentState1 = 0;

int subcurrentState1 = 0;

int addpreviousState1=1;

int subpreviousState1=1;

int lane1=3;

///////////////////LANE 2

int IRinLane2 = 24;

int IRoutLane2 = 25;

int addcurrentState2 = 0;

int subcurrentState2 = 0;

int addpreviousState2=1;

int subpreviousState2=1;

int lane2=1;

///////////////////LANE 3

int IRinLane3 = 26;

int IRoutLane3 = 27;

int addcurrentState3 = 0;

int subcurrentState3 = 0;

int addpreviousState3=1;

int subpreviousState3=1;

int lane3=1;

///////////////////LANE 4

int IRinLane4 = 28;

int IRoutLane4 = 29;

int addcurrentState4 = 0;

int subcurrentState4 = 0;

int addpreviousState4=1;

int subpreviousState4=1;

int lane4=1;

int checkSensor1 (){

  addcurrentState1 = digitalRead(IRinLane1);

 subcurrentState1 = digitalRead(IRoutLane1);

/////////////////////////////////////////////////////InlaneA

 if(addcurrentState1!=addpreviousState1){

  if(addcurrentState1 == 0){

   lane1++;}

   if(lane1 > 5)

     lane1=5;

 }

 addpreviousState1 = addcurrentState1; ///update

////////////////////////////////////////////////////OutlaneA

 if(subcurrentState1!=subpreviousState1){

  if(subcurrentState1 == 0){

   lane1--;}

   if(lane1 < 0)

     lane1=0;

 }

 subpreviousState1 = subcurrentState1;

//Serial.print ("Lane A: ");

//Serial.println(lane1);

}

int checkSensor2 (){

   addcurrentState2 = digitalRead(IRinLane2);

 subcurrentState2 = digitalRead(IRoutLane2);

/////////////////////////////////////////////////////InlaneB

 if(addcurrentState2!=addpreviousState2){

  if(addcurrentState2 == 0){

   lane2++;}

   if(lane2 > 5)

     lane2=5;

 }

 addpreviousState2 = addcurrentState2; ///update

////////////////////////////////////////////////////OutlaneB

 if(subcurrentState2!=subpreviousState2){

  if(subcurrentState2 == 0){

   lane2--;}

   if(lane2 < 0)

     lane2=0;

 }

 subpreviousState2 = subcurrentState2;

 //Serial.print ("Lane B: ");

 //Serial.println(lane2);

}

int checkSensor3 (){

   addcurrentState3 = digitalRead(IRinLane3);

 subcurrentState3 = digitalRead(IRoutLane3);

/////////////////////////////////////////////////////InlaneC

 if(addcurrentState3!=addpreviousState3){

  if(addcurrentState3 == 0){

   lane3++;}

   if(lane3 > 5)

     lane3=5;

 }

 addpreviousState3 = addcurrentState3; ///update

////////////////////////////////////////////////////OutlaneC

 if(subcurrentState3!=subpreviousState3){

  if(subcurrentState3 == 0){

   lane3--;}

   if(lane3 < 0)

     lane3=0;

 }

 subpreviousState3 = subcurrentState3;

 //Serial.print ("Lane C: ");

 //Serial.println(lane3);

}

int checkSensor4 (){

   addcurrentState4 = digitalRead(IRinLane4);

 subcurrentState4 = digitalRead(IRoutLane4);

/////////////////////////////////////////////////////InlaneD

 if(addcurrentState4!=addpreviousState4){

  if(addcurrentState4 == 0){

   lane4++;}

   if(lane4 > 5)

     lane4=5;

 }

 addpreviousState4 = addcurrentState4; ///update

////////////////////////////////////////////////////OutlaneD

 if(subcurrentState4!=subpreviousState4){

  if(subcurrentState4 == 0){

   lane4--;}

   if(lane4 < 0)

     lane4=0;

 }

 subpreviousState4 = subcurrentState4;

 //Serial.print ("Lane D: ");

 //Serial.println(lane4);

}

void setup() {

**Serial**.begin(57600);

 pinMode(IRinLane1 , INPUT);

 pinMode(IRoutLane1 , INPUT);

 pinMode(IRinLane2 , INPUT);

 pinMode(IRoutLane2 , INPUT);

 pinMode(IRinLane3 , INPUT);

 pinMode(IRoutLane3 , INPUT);

 pinMode(IRinLane4 , INPUT);

 pinMode(IRoutLane4 , INPUT);

 pinMode (rA, OUTPUT);

 pinMode (yA, OUTPUT);

 pinMode (gA, OUTPUT);

 pinMode (rB, OUTPUT);

 pinMode (yB, OUTPUT);

 pinMode (gB, OUTPUT);

 pinMode (rC, OUTPUT);

 pinMode (yC, OUTPUT);

 pinMode (gC, OUTPUT);

 pinMode (rD, OUTPUT);

 pinMode (yD, OUTPUT);

 pinMode (gD, OUTPUT);

}

// Trapezoidal Member Function

FIS\_TYPE fis\_trapmf(FIS\_TYPE x, FIS\_TYPE\* p)

{

   FIS\_TYPE a = p[0], b = p[1], c = p[2], d = p[3];

   FIS\_TYPE t1 = ((x <= c) ? 1 : ((d < x) ? 0 : ((c != d) ? ((d - x) / (d - c)) : 0)));

   FIS\_TYPE t2 = ((b <= x) ? 1 : ((x < a) ? 0 : ((a != b) ? ((x - a) / (b - a)) : 0)));

   return (FIS\_TYPE) min(t1, t2);

}

// Gaussian Member Function

FIS\_TYPE fis\_gaussmf(FIS\_TYPE x, FIS\_TYPE\* p)

{

   FIS\_TYPE s = p[0], c = p[1];

   FIS\_TYPE t = (x - c) / s;

   return exp(-(t \* t) / 2);

}

FIS\_TYPE fis\_min(FIS\_TYPE a, FIS\_TYPE b)

{

   return min(a, b);

}

FIS\_TYPE fis\_max(FIS\_TYPE a, FIS\_TYPE b)

{

   return max(a, b);

}

FIS\_TYPE fis\_array\_operation(FIS\_TYPE \*array, int size, \_FIS\_ARR\_OP pfnOp)

{

   int i;

   FIS\_TYPE ret = 0;

   if (size == 0) return ret;

   if (size == 1) return array[0];

   ret = array[0];

   for (i = 1; i < size; i++)

   {

       ret = (\*pfnOp)(ret, array[i]);

   }

   return ret;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Data for Fuzzy Inference System

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Pointers to the implementations of member functions

\_FIS\_MF fis\_gMF[] =

{

   fis\_trapmf, fis\_gaussmf

};

// Count of member function for each Input

int fis\_gIMFCount[] = { 3, 3, 3, 3 };

// Count of member function for each Output

int fis\_gOMFCount[] = { 24 };

// Coefficients for the Input Member Functions

FIS\_TYPE fis\_gMFI0Coeff1[] = { -0.5, 0, 1, 1.5 };

FIS\_TYPE fis\_gMFI0Coeff2[] = { 1.3, 2, 3, 3.5 };

FIS\_TYPE fis\_gMFI0Coeff3[] = { 3.3, 4, 5, 5.5 };

FIS\_TYPE\* fis\_gMFI0Coeff[] = { fis\_gMFI0Coeff1, fis\_gMFI0Coeff2, fis\_gMFI0Coeff3 };

FIS\_TYPE fis\_gMFI1Coeff1[] = { -0.5, 0, 1, 1.5 };

FIS\_TYPE fis\_gMFI1Coeff2[] = { 1.3, 2, 3, 3.5 };

FIS\_TYPE fis\_gMFI1Coeff3[] = { 3.3, 4, 5, 5.5 };

FIS\_TYPE\* fis\_gMFI1Coeff[] = { fis\_gMFI1Coeff1, fis\_gMFI1Coeff2, fis\_gMFI1Coeff3 };

FIS\_TYPE fis\_gMFI2Coeff1[] = { -0.5, 0, 1, 1.5 };

FIS\_TYPE fis\_gMFI2Coeff2[] = { 1.3, 2, 3, 3.5 };

FIS\_TYPE fis\_gMFI2Coeff3[] = { 3.3, 4, 5, 5.5 };

FIS\_TYPE\* fis\_gMFI2Coeff[] = { fis\_gMFI2Coeff1, fis\_gMFI2Coeff2, fis\_gMFI2Coeff3 };

FIS\_TYPE fis\_gMFI3Coeff1[] = { -0.5, 0, 1, 1.5 };

FIS\_TYPE fis\_gMFI3Coeff2[] = { 1.3, 2, 3, 3.5 };

FIS\_TYPE fis\_gMFI3Coeff3[] = { 3.3, 4, 5, 5.5 };

FIS\_TYPE\* fis\_gMFI3Coeff[] = { fis\_gMFI3Coeff1, fis\_gMFI3Coeff2, fis\_gMFI3Coeff3 };

FIS\_TYPE\*\* fis\_gMFICoeff[] = { fis\_gMFI0Coeff, fis\_gMFI1Coeff, fis\_gMFI2Coeff, fis\_gMFI3Coeff };

// Coefficients for the Output Member Functions

FIS\_TYPE fis\_gMFO0Coeff1[] = { 0.5, 1 };

FIS\_TYPE fis\_gMFO0Coeff2[] = { 0.5, 2 };

FIS\_TYPE fis\_gMFO0Coeff3[] = { 0.5, 3 };

FIS\_TYPE fis\_gMFO0Coeff4[] = { 0.5, 4 };

FIS\_TYPE fis\_gMFO0Coeff5[] = { 0.5, 5 };

FIS\_TYPE fis\_gMFO0Coeff6[] = { 0.5, 6 };

FIS\_TYPE fis\_gMFO0Coeff7[] = { 0.5, 7 };

FIS\_TYPE fis\_gMFO0Coeff8[] = { 0.5, 8 };

FIS\_TYPE fis\_gMFO0Coeff9[] = { 0.5, 9 };

FIS\_TYPE fis\_gMFO0Coeff10[] = { 0.5, 10 };

FIS\_TYPE fis\_gMFO0Coeff11[] = { 0.5, 11 };

FIS\_TYPE fis\_gMFO0Coeff12[] = { 0.5, 12 };

FIS\_TYPE fis\_gMFO0Coeff13[] = { 0.5, 13 };

FIS\_TYPE fis\_gMFO0Coeff14[] = { 0.5, 14 };

FIS\_TYPE fis\_gMFO0Coeff15[] = { 0.5, 15 };

FIS\_TYPE fis\_gMFO0Coeff16[] = { 0.5, 16 };

FIS\_TYPE fis\_gMFO0Coeff17[] = { 0.5, 17 };

FIS\_TYPE fis\_gMFO0Coeff18[] = { 0.5, 18 };

FIS\_TYPE fis\_gMFO0Coeff19[] = { 0.5, 19 };

FIS\_TYPE fis\_gMFO0Coeff20[] = { 0.5, 20 };

FIS\_TYPE fis\_gMFO0Coeff21[] = { 0.5, 21 };

FIS\_TYPE fis\_gMFO0Coeff22[] = { 0.5, 22 };

FIS\_TYPE fis\_gMFO0Coeff23[] = { 0.5, 23 };

FIS\_TYPE fis\_gMFO0Coeff24[] = { 0.5, 24 };

FIS\_TYPE\* fis\_gMFO0Coeff[] = { fis\_gMFO0Coeff1, fis\_gMFO0Coeff2, fis\_gMFO0Coeff3, fis\_gMFO0Coeff4, fis\_gMFO0Coeff5, fis\_gMFO0Coeff6, fis\_gMFO0Coeff7, fis\_gMFO0Coeff8, fis\_gMFO0Coeff9, fis\_gMFO0Coeff10, fis\_gMFO0Coeff11, fis\_gMFO0Coeff12, fis\_gMFO0Coeff13, fis\_gMFO0Coeff14, fis\_gMFO0Coeff15, fis\_gMFO0Coeff16, fis\_gMFO0Coeff17, fis\_gMFO0Coeff18, fis\_gMFO0Coeff19, fis\_gMFO0Coeff20, fis\_gMFO0Coeff21, fis\_gMFO0Coeff22, fis\_gMFO0Coeff23, fis\_gMFO0Coeff24 };

FIS\_TYPE\*\* fis\_gMFOCoeff[] = { fis\_gMFO0Coeff };

// Input membership function set

int fis\_gMFI0[] = { 0, 0, 0 };

int fis\_gMFI1[] = { 0, 0, 0 };

int fis\_gMFI2[] = { 0, 0, 0 };

int fis\_gMFI3[] = { 0, 0, 0 };

int\* fis\_gMFI[] = { fis\_gMFI0, fis\_gMFI1, fis\_gMFI2, fis\_gMFI3};

// Output membership function set

int fis\_gMFO0[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

int\* fis\_gMFO[] = { fis\_gMFO0};

// Rule Weights

FIS\_TYPE fis\_gRWeight[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Type

int fis\_gRType[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Inputs

int fis\_gRI0[] = { 3, 3, 3, 3 };

int fis\_gRI1[] = { 3, 3, 3, 2 };

int fis\_gRI2[] = { 3, 3, 3, 1 };

int fis\_gRI3[] = { 3, 3, 2, 2 };

int fis\_gRI4[] = { 3, 3, 2, 1 };

int fis\_gRI5[] = { 3, 3, 1, 1 };

int fis\_gRI6[] = { 3, 2, 2, 2 };

int fis\_gRI7[] = { 3, 2, 2, 1 };

int fis\_gRI8[] = { 3, 2, 1, 1 };

int fis\_gRI9[] = { 3, 1, 1, 1 };

int fis\_gRI10[] = { 2, 2, 2, 2 };

int fis\_gRI11[] = { 2, 2, 2, 1 };

int fis\_gRI12[] = { 2, 2, 1, 1 };

int fis\_gRI13[] = { 2, 1, 1, 1 };

int fis\_gRI14[] = { 1, 1, 1, 1 };

int fis\_gRI15[] = { 3, 3, 2, 3 };

int fis\_gRI16[] = { 3, 3, 1, 3 };

int fis\_gRI17[] = { 3, 3, 1, 2 };

int fis\_gRI18[] = { 3, 2, 1, 2 };

int fis\_gRI19[] = { 2, 2, 1, 2 };

int fis\_gRI20[] = { 3, 2, 3, 2 };

int fis\_gRI21[] = { 3, 2, 3, 1 };

int fis\_gRI22[] = { 3, 1, 3, 1 };

int fis\_gRI23[] = { 3, 1, 2, 1 };

int fis\_gRI24[] = { 2, 1, 2, 1 };

int fis\_gRI25[] = { 3, 2, 3, 3 };

int fis\_gRI26[] = { 3, 1, 3, 3 };

int fis\_gRI27[] = { 3, 1, 3, 2 };

int fis\_gRI28[] = { 3, 1, 2, 2 };

int fis\_gRI29[] = { 2, 1, 2, 2 };

int fis\_gRI30[] = { 3, 2, 2, 3 };

int fis\_gRI31[] = { 3, 2, 1, 3 };

int fis\_gRI32[] = { 3, 1, 1, 3 };

int fis\_gRI33[] = { 3, 1, 1, 2 };

int fis\_gRI34[] = { 2, 1, 1, 2 };

int fis\_gRI35[] = { 3, 1, 2, 3 };

int fis\_gRI36[] = { 2, 3, 2, 2 };

int fis\_gRI37[] = { 2, 3, 2, 1 };

int fis\_gRI38[] = { 2, 3, 1, 1 };

int fis\_gRI39[] = { 1, 3, 1, 1 };

int fis\_gRI40[] = { 1, 2, 1, 1 };

int fis\_gRI41[] = { 2, 3, 1, 2 };

int fis\_gRI42[] = { 2, 3, 3, 2 };

int fis\_gRI43[] = { 2, 3, 3, 1 };

int fis\_gRI44[] = { 1, 3, 3, 1 };

int fis\_gRI45[] = { 1, 3, 2, 1 };

int fis\_gRI46[] = { 1, 2, 2, 1 };

int fis\_gRI47[] = { 2, 3, 3, 3 };

int fis\_gRI48[] = { 1, 3, 3, 3 };

int fis\_gRI49[] = { 1, 3, 3, 2 };

int fis\_gRI50[] = { 1, 3, 2, 2 };

int fis\_gRI51[] = { 1, 2, 2, 2 };

int fis\_gRI52[] = { 2, 3, 2, 3 };

int fis\_gRI53[] = { 2, 3, 1, 3 };

int fis\_gRI54[] = { 1, 3, 1, 3 };

int fis\_gRI55[] = { 1, 3, 1, 2 };

int fis\_gRI56[] = { 1, 2, 1, 2 };

int fis\_gRI57[] = { 1, 3, 2, 3 };

int fis\_gRI58[] = { 2, 2, 3, 2 };

int fis\_gRI59[] = { 2, 2, 3, 1 };

int fis\_gRI60[] = { 2, 1, 3, 1 };

int fis\_gRI61[] = { 1, 1, 3, 1 };

int fis\_gRI62[] = { 1, 1, 2, 1 };

int fis\_gRI63[] = { 2, 1, 3, 2 };

int fis\_gRI64[] = { 1, 2, 3, 1 };

int fis\_gRI65[] = { 1, 2, 3, 2 };

int fis\_gRI66[] = { 2, 2, 3, 3 };

int fis\_gRI67[] = { 2, 1, 3, 3 };

int fis\_gRI68[] = { 1, 1, 3, 3 };

int fis\_gRI69[] = { 1, 1, 3, 2 };

int fis\_gRI70[] = { 1, 1, 2, 2 };

int fis\_gRI71[] = { 1, 2, 3, 3 };

int fis\_gRI72[] = { 2, 2, 2, 3 };

int fis\_gRI73[] = { 2, 2, 1, 3 };

int fis\_gRI74[] = { 2, 1, 1, 3 };

int fis\_gRI75[] = { 1, 1, 1, 3 };

int fis\_gRI76[] = { 1, 1, 1, 2 };

int fis\_gRI77[] = { 2, 1, 2, 3 };

int fis\_gRI78[] = { 1, 2, 1, 3 };

int fis\_gRI79[] = { 1, 2, 2, 3 };

int fis\_gRI80[] = { 1, 1, 2, 3 };

int\* fis\_gRI[] = { fis\_gRI0, fis\_gRI1, fis\_gRI2, fis\_gRI3, fis\_gRI4, fis\_gRI5, fis\_gRI6, fis\_gRI7, fis\_gRI8, fis\_gRI9, fis\_gRI10, fis\_gRI11, fis\_gRI12, fis\_gRI13, fis\_gRI14, fis\_gRI15, fis\_gRI16, fis\_gRI17, fis\_gRI18, fis\_gRI19, fis\_gRI20, fis\_gRI21, fis\_gRI22, fis\_gRI23, fis\_gRI24, fis\_gRI25, fis\_gRI26, fis\_gRI27, fis\_gRI28, fis\_gRI29, fis\_gRI30, fis\_gRI31, fis\_gRI32, fis\_gRI33, fis\_gRI34, fis\_gRI35, fis\_gRI36, fis\_gRI37, fis\_gRI38, fis\_gRI39, fis\_gRI40, fis\_gRI41, fis\_gRI42, fis\_gRI43, fis\_gRI44, fis\_gRI45, fis\_gRI46, fis\_gRI47, fis\_gRI48, fis\_gRI49, fis\_gRI50, fis\_gRI51, fis\_gRI52, fis\_gRI53, fis\_gRI54, fis\_gRI55, fis\_gRI56, fis\_gRI57, fis\_gRI58, fis\_gRI59, fis\_gRI60, fis\_gRI61, fis\_gRI62, fis\_gRI63, fis\_gRI64, fis\_gRI65, fis\_gRI66, fis\_gRI67, fis\_gRI68, fis\_gRI69, fis\_gRI70, fis\_gRI71, fis\_gRI72, fis\_gRI73, fis\_gRI74, fis\_gRI75, fis\_gRI76, fis\_gRI77, fis\_gRI78, fis\_gRI79, fis\_gRI80 };

// Rule Outputs

int fis\_gRO0[] = { 1 };

int fis\_gRO1[] = { 1 };

int fis\_gRO2[] = { 1 };

int fis\_gRO3[] = { 1 };

int fis\_gRO4[] = { 1 };

int fis\_gRO5[] = { 1 };

int fis\_gRO6[] = { 1 };

int fis\_gRO7[] = { 1 };

int fis\_gRO8[] = { 1 };

int fis\_gRO9[] = { 1 };

int fis\_gRO10[] = { 1 };

int fis\_gRO11[] = { 1 };

int fis\_gRO12[] = { 1 };

int fis\_gRO13[] = { 1 };

int fis\_gRO14[] = { 1 };

int fis\_gRO15[] = { 2 };

int fis\_gRO16[] = { 2 };

int fis\_gRO17[] = { 2 };

int fis\_gRO18[] = { 2 };

int fis\_gRO19[] = { 2 };

int fis\_gRO20[] = { 3 };

int fis\_gRO21[] = { 3 };

int fis\_gRO22[] = { 3 };

int fis\_gRO23[] = { 3 };

int fis\_gRO24[] = { 3 };

int fis\_gRO25[] = { 4 };

int fis\_gRO26[] = { 4 };

int fis\_gRO27[] = { 4 };

int fis\_gRO28[] = { 4 };

int fis\_gRO29[] = { 4 };

int fis\_gRO30[] = { 5 };

int fis\_gRO31[] = { 5 };

int fis\_gRO32[] = { 5 };

int fis\_gRO33[] = { 5 };

int fis\_gRO34[] = { 5 };

int fis\_gRO35[] = { 6 };

int fis\_gRO36[] = { 7 };

int fis\_gRO37[] = { 7 };

int fis\_gRO38[] = { 7 };

int fis\_gRO39[] = { 7 };

int fis\_gRO40[] = { 7 };

int fis\_gRO41[] = { 8 };

int fis\_gRO42[] = { 9 };

int fis\_gRO43[] = { 9 };

int fis\_gRO44[] = { 9 };

int fis\_gRO45[] = { 9 };

int fis\_gRO46[] = { 9 };

int fis\_gRO47[] = { 10 };

int fis\_gRO48[] = { 10 };

int fis\_gRO49[] = { 10 };

int fis\_gRO50[] = { 10 };

int fis\_gRO51[] = { 10 };

int fis\_gRO52[] = { 11 };

int fis\_gRO53[] = { 11 };

int fis\_gRO54[] = { 11 };

int fis\_gRO55[] = { 11 };

int fis\_gRO56[] = { 11 };

int fis\_gRO57[] = { 12 };

int fis\_gRO58[] = { 13 };

int fis\_gRO59[] = { 13 };

int fis\_gRO60[] = { 13 };

int fis\_gRO61[] = { 13 };

int fis\_gRO62[] = { 13 };

int fis\_gRO63[] = { 14 };

int fis\_gRO64[] = { 15 };

int fis\_gRO65[] = { 16 };

int fis\_gRO66[] = { 17 };

int fis\_gRO67[] = { 17 };

int fis\_gRO68[] = { 17 };

int fis\_gRO69[] = { 17 };

int fis\_gRO70[] = { 17 };

int fis\_gRO71[] = { 18 };

int fis\_gRO72[] = { 19 };

int fis\_gRO73[] = { 19 };

int fis\_gRO74[] = { 19 };

int fis\_gRO75[] = { 19 };

int fis\_gRO76[] = { 19 };

int fis\_gRO77[] = { 20 };

int fis\_gRO78[] = { 21 };

int fis\_gRO79[] = { 22 };

int fis\_gRO80[] = { 23 };

int\* fis\_gRO[] = { fis\_gRO0, fis\_gRO1, fis\_gRO2, fis\_gRO3, fis\_gRO4, fis\_gRO5, fis\_gRO6, fis\_gRO7, fis\_gRO8, fis\_gRO9, fis\_gRO10, fis\_gRO11, fis\_gRO12, fis\_gRO13, fis\_gRO14, fis\_gRO15, fis\_gRO16, fis\_gRO17, fis\_gRO18, fis\_gRO19, fis\_gRO20, fis\_gRO21, fis\_gRO22, fis\_gRO23, fis\_gRO24, fis\_gRO25, fis\_gRO26, fis\_gRO27, fis\_gRO28, fis\_gRO29, fis\_gRO30, fis\_gRO31, fis\_gRO32, fis\_gRO33, fis\_gRO34, fis\_gRO35, fis\_gRO36, fis\_gRO37, fis\_gRO38, fis\_gRO39, fis\_gRO40, fis\_gRO41, fis\_gRO42, fis\_gRO43, fis\_gRO44, fis\_gRO45, fis\_gRO46, fis\_gRO47, fis\_gRO48, fis\_gRO49, fis\_gRO50, fis\_gRO51, fis\_gRO52, fis\_gRO53, fis\_gRO54, fis\_gRO55, fis\_gRO56, fis\_gRO57, fis\_gRO58, fis\_gRO59, fis\_gRO60, fis\_gRO61, fis\_gRO62, fis\_gRO63, fis\_gRO64, fis\_gRO65, fis\_gRO66, fis\_gRO67, fis\_gRO68, fis\_gRO69, fis\_gRO70, fis\_gRO71, fis\_gRO72, fis\_gRO73, fis\_gRO74, fis\_gRO75, fis\_gRO76, fis\_gRO77, fis\_gRO78, fis\_gRO79, fis\_gRO80 };

// Input range Min

FIS\_TYPE fis\_gIMin[] = { 0, 0, 0, 0 };

// Input range Max

FIS\_TYPE fis\_gIMax[] = { 5, 5, 5, 5 };

// Output range Min

FIS\_TYPE fis\_gOMin[] = { 0 };

// Output range Max

FIS\_TYPE fis\_gOMax[] = { 24 };

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Data dependent support functions for Fuzzy Inference System

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FIS\_TYPE fis\_MF\_out(FIS\_TYPE\*\* fuzzyRuleSet, FIS\_TYPE x, int o)

{

   FIS\_TYPE mfOut;

   int r;

   for (r = 0; r < fis\_gcR; ++r)

   {

       int index = fis\_gRO[r][o];

       if (index > 0)

       {

           index = index - 1;

           mfOut = (fis\_gMF[fis\_gMFO[o][index]])(x, fis\_gMFOCoeff[o][index]);

       }

       else if (index < 0)

       {

           index = -index - 1;

           mfOut = 1 - (fis\_gMF[fis\_gMFO[o][index]])(x, fis\_gMFOCoeff[o][index]);

       }

       else

       {

           mfOut = 0;

       }

       fuzzyRuleSet[0][r] = fis\_min(mfOut, fuzzyRuleSet[1][r]);

   }

   return fis\_array\_operation(fuzzyRuleSet[0], fis\_gcR, fis\_max);

}

FIS\_TYPE fis\_defuzz\_centroid(FIS\_TYPE\*\* fuzzyRuleSet, int o)

{

   FIS\_TYPE step = (fis\_gOMax[o] - fis\_gOMin[o]) / (FIS\_RESOLUSION - 1);

   FIS\_TYPE area = 0;

   FIS\_TYPE momentum = 0;

   FIS\_TYPE dist, slice;

   int i;

   // calculate the area under the curve formed by the MF outputs

   for (i = 0; i < FIS\_RESOLUSION; ++i){

       dist = fis\_gOMin[o] + (step \* i);

       slice = step \* fis\_MF\_out(fuzzyRuleSet, dist, o);

       area += slice;

       momentum += slice\*dist;

   }

   return ((area == 0) ? ((fis\_gOMax[o] + fis\_gOMin[o]) / 2) : (momentum / area));

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Fuzzy Inference System

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void fis\_evaluate()

{

   FIS\_TYPE fuzzyInput0[] = { 0, 0, 0 };

   FIS\_TYPE fuzzyInput1[] = { 0, 0, 0 };

   FIS\_TYPE fuzzyInput2[] = { 0, 0, 0 };

   FIS\_TYPE fuzzyInput3[] = { 0, 0, 0 };

   FIS\_TYPE\* fuzzyInput[fis\_gcI] = { fuzzyInput0, fuzzyInput1, fuzzyInput2, fuzzyInput3, };

   FIS\_TYPE fuzzyOutput0[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

   FIS\_TYPE\* fuzzyOutput[fis\_gcO] = { fuzzyOutput0, };

   FIS\_TYPE fuzzyRules[fis\_gcR] = { 0 };

   FIS\_TYPE fuzzyFires[fis\_gcR] = { 0 };

   FIS\_TYPE\* fuzzyRuleSet[] = { fuzzyRules, fuzzyFires };

   FIS\_TYPE sW = 0;

   // Transforming input to fuzzy Input

   int i, j, r, o;

   for (i = 0; i < fis\_gcI; ++i)

   {

       for (j = 0; j < fis\_gIMFCount[i]; ++j)

       {

           fuzzyInput[i][j] =

               (fis\_gMF[fis\_gMFI[i][j]])(g\_fisInput[i], fis\_gMFICoeff[i][j]);

       }

   }

   int index = 0;

   for (r = 0; r < fis\_gcR; ++r)

   {

       if (fis\_gRType[r] == 1)

       {

           fuzzyFires[r] = FIS\_MAX;

           for (i = 0; i < fis\_gcI; ++i)

           {

               index = fis\_gRI[r][i];

               if (index > 0)

                   fuzzyFires[r] = fis\_min(fuzzyFires[r], fuzzyInput[i][index - 1]);

               else if (index < 0)

                   fuzzyFires[r] = fis\_min(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);

               else

                   fuzzyFires[r] = fis\_min(fuzzyFires[r], 1);

           }

       }

       else

       {

           fuzzyFires[r] = FIS\_MIN;

           for (i = 0; i < fis\_gcI; ++i)

           {

               index = fis\_gRI[r][i];

               if (index > 0)

                   fuzzyFires[r] = fis\_max(fuzzyFires[r], fuzzyInput[i][index - 1]);

               else if (index < 0)

                   fuzzyFires[r] = fis\_max(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);

               else

                   fuzzyFires[r] = fis\_max(fuzzyFires[r], 0);

           }

       }

       fuzzyFires[r] = fis\_gRWeight[r] \* fuzzyFires[r];

       sW += fuzzyFires[r];

   }

   if (sW == 0)

   {

       for (o = 0; o < fis\_gcO; ++o)

       {

           g\_fisOutput[o] = ((fis\_gOMax[o] + fis\_gOMin[o]) / 2);

       }

   }

   else

   {

       for (o = 0; o < fis\_gcO; ++o)

       {

           g\_fisOutput[o] = fis\_defuzz\_centroid(fuzzyRuleSet, o);

       }

   }

}

// Trapezoidal Member Function

FIS1\_TYPE fis1\_trapmf(FIS1\_TYPE y, FIS1\_TYPE\* q)

{

   FIS1\_TYPE a1 = q[0], b1 = q[1], c1 = q[2], d1 = q[3];

   FIS1\_TYPE t11 = ((y <= c1) ? 1 : ((d1 < y) ? 0 : ((c1 != d1) ? ((d1 - y) / (d1 - c1)) : 0)));

   FIS1\_TYPE t22 = ((b1 <= y) ? 1 : ((y < a1) ? 0 : ((a1 != b1) ? ((y - a1) / (b1 - a1)) : 0)));

   return (FIS1\_TYPE) min(t11, t22);

}

FIS1\_TYPE fis1\_prod(FIS1\_TYPE a1, FIS1\_TYPE b1)

{

   return (a1 \* b1);

}

FIS1\_TYPE fis1\_probor(FIS1\_TYPE a1, FIS1\_TYPE b1)

{

   return (a1 + b1 - (a1 \* b1));

}

FIS1\_TYPE fis1\_sum(FIS1\_TYPE a1, FIS1\_TYPE b1)

{

   return (a1 + b1);

}

FIS1\_TYPE fis1\_array\_operation(FIS1\_TYPE \*array1, int size1, \_FIS1\_ARR\_OP pfnOp1)

{

   int i1;

   FIS1\_TYPE ret1 = 0;

   if (size1 == 0) return ret1;

   if (size1 == 1) return array1[0];

   ret1 = array1[0];

   for (i1 = 1; i1 < size1; i1++)

   {

       ret1 = (\*pfnOp1)(ret1, array1[i1]);

   }

   return ret1;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Data for Fuzzy Inference System

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Pointers to the implementations of member functions

\_FIS1\_MF fis1\_gMF[] =

{

   fis1\_trapmf

};

// Count of member function for each Input

int fis1\_gIMFCount[] = { 5, 5 };

// Count of member function for each Output

int fis1\_gOMFCount[] = { 25 };

// Coefficients for the Input Member Functions

FIS1\_TYPE fis1\_gMFI0Coeff1[] = { -0.875, -0.375, 0.375, 0.875 };

FIS1\_TYPE fis1\_gMFI0Coeff2[] = { 0.375, 0.875, 1.64056799354297, 2.16459321862527 };

FIS1\_TYPE fis1\_gMFI0Coeff3[] = { 1.63063835733303, 2.14080805819318, 2.85899037882654, 3.36928490659435 };

FIS1\_TYPE fis1\_gMFI0Coeff4[] = { 2.83497677849296, 3.35923574110981, 4.125, 4.625 };

FIS1\_TYPE fis1\_gMFI0Coeff5[] = { 4.125, 4.625, 5.375, 5.875 };

FIS1\_TYPE\* fis1\_gMFI0Coeff[] = { fis1\_gMFI0Coeff1, fis1\_gMFI0Coeff2, fis1\_gMFI0Coeff3, fis1\_gMFI0Coeff4, fis1\_gMFI0Coeff5 };

FIS1\_TYPE fis1\_gMFI1Coeff1[] = { -2.625, -1.125, 1.12505222700691, 2.62507311169085 };

FIS1\_TYPE fis1\_gMFI1Coeff2[] = { 1.12503730730547, 2.62505222710532, 4.81733464189828, 6.02673467764801 };

FIS1\_TYPE fis1\_gMFI1Coeff3[] = { 5.001, 6.318, 8.618, 10.13 };

FIS1\_TYPE fis1\_gMFI1Coeff4[] = { 8.60162311400026, 10.1176155756374, 12.3681881262356, 13.8700948418865 };

FIS1\_TYPE fis1\_gMFI1Coeff5[] = { 12.3655719191175, 13.8681897609811, 16.125, 17.625 };

FIS1\_TYPE\* fis1\_gMFI1Coeff[] = { fis1\_gMFI1Coeff1, fis1\_gMFI1Coeff2, fis1\_gMFI1Coeff3, fis1\_gMFI1Coeff4, fis1\_gMFI1Coeff5 };

FIS1\_TYPE\*\* fis1\_gMFICoeff[] = { fis1\_gMFI0Coeff, fis1\_gMFI1Coeff };

// Coefficients for the Output Member Functions

FIS\_TYPE fis1\_gMFO0Coeff1[] = { 0, 0, 0 };

FIS\_TYPE fis1\_gMFO0Coeff2[] = { 0, 0, 0 };

FIS\_TYPE fis1\_gMFO0Coeff3[] = { 0, 0, 0 };

FIS\_TYPE fis1\_gMFO0Coeff4[] = { 0, 0, 0 };

FIS\_TYPE fis1\_gMFO0Coeff5[] = { 0, 0, 0 };

FIS\_TYPE fis1\_gMFO0Coeff6[] = { 11.6245142079824, -0.000516303325366294, 8.37565061584225 };

FIS\_TYPE fis1\_gMFO0Coeff7[] = { 11.62503848856, -0.000902869795866519, 8.37813403214309 };

FIS\_TYPE fis1\_gMFO0Coeff8[] = { 8.12693457320223, -0.0996814242280877, 3.93360545386292 };

FIS\_TYPE fis1\_gMFO0Coeff9[] = { 7.75800210904754, -0.0714460227440342, 3.9378010120765 };

FIS\_TYPE fis1\_gMFO0Coeff10[] = { 7.54783143274267, -0.0504065864216953, 3.87404604478669 };

FIS\_TYPE fis1\_gMFO0Coeff11[] = { 11.4852573039919, 0.000186761397374774, 6.32466640270123 };

FIS\_TYPE fis1\_gMFO0Coeff12[] = { 11.4852873008064, 0.000313103497599302, 6.3231711115608 };

FIS\_TYPE fis1\_gMFO0Coeff13[] = { 10.259449366656, -0.409931533412153, 6.6417388943016 };

FIS\_TYPE fis1\_gMFO0Coeff14[] = { 9.69080625188677, -0.212350175654589, 6.16550022838491 };

FIS\_TYPE fis1\_gMFO0Coeff15[] = { 9.43423399916889, -0.14393594370801, 5.96747840269359 };

FIS\_TYPE fis1\_gMFO0Coeff16[] = { 11.7944211378877, 0.000888225384641451, 2.8220210226435 };

FIS\_TYPE fis1\_gMFO0Coeff17[] = { 11.7931519202651, 0.00152799826231328, 2.82203218489859 };

FIS\_TYPE fis1\_gMFO0Coeff18[] = { 12.036340198493, -0.720478649290368, 2.68024521081998 };

FIS\_TYPE fis1\_gMFO0Coeff19[] = { 11.2607991180657, -0.35344224017186, 2.52809895965356 };

FIS\_TYPE fis1\_gMFO0Coeff20[] = { 10.938467846828, -0.237710468770196, 2.46268258392751 };

FIS\_TYPE fis1\_gMFO0Coeff21[] = { 11.5383746121552, 0.00135131493556646, 2.30767492785326 };

FIS\_TYPE fis1\_gMFO0Coeff22[] = { 11.5369024168278, 0.00232492636206018, 2.30738048586132 };

FIS\_TYPE fis1\_gMFO0Coeff23[] = { 12.253026769109, -0.923062597240521, 2.45060535777194 };

FIS\_TYPE fis1\_gMFO0Coeff24[] = { 11.443643558954, -0.445454885172979, 2.28872870949715 };

FIS\_TYPE fis1\_gMFO0Coeff25[] = { 11.1144469743303, -0.298801098949864, 2.22288939356804 };

FIS1\_TYPE\* fis1\_gMFO0Coeff[] = { fis1\_gMFO0Coeff1, fis1\_gMFO0Coeff2, fis1\_gMFO0Coeff3, fis1\_gMFO0Coeff4, fis1\_gMFO0Coeff5, fis1\_gMFO0Coeff6, fis1\_gMFO0Coeff7, fis1\_gMFO0Coeff8, fis1\_gMFO0Coeff9, fis1\_gMFO0Coeff10, fis1\_gMFO0Coeff11, fis1\_gMFO0Coeff12, fis1\_gMFO0Coeff13, fis1\_gMFO0Coeff14, fis1\_gMFO0Coeff15, fis1\_gMFO0Coeff16, fis1\_gMFO0Coeff17, fis1\_gMFO0Coeff18, fis1\_gMFO0Coeff19, fis1\_gMFO0Coeff20, fis1\_gMFO0Coeff21, fis1\_gMFO0Coeff22, fis1\_gMFO0Coeff23, fis1\_gMFO0Coeff24, fis1\_gMFO0Coeff25 };

FIS1\_TYPE\*\* fis1\_gMFOCoeff[] = { fis1\_gMFO0Coeff };

// Input membership function set

int fis1\_gMFI0[] = { 0, 0, 0, 0, 0 };

int fis1\_gMFI1[] = { 0, 0, 0, 0, 0 };

int\* fis1\_gMFI[] = { fis1\_gMFI0, fis1\_gMFI1};

// Output membership function set

int\* fis1\_gMFO[] = {};

// Rule Weights

FIS1\_TYPE fis1\_gRWeight[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Type

int fis1\_gRType[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Inputs

int fis1\_gRI0[] = { 1, 1 };

int fis1\_gRI1[] = { 1, 2 };

int fis1\_gRI2[] = { 1, 3 };

int fis1\_gRI3[] = { 1, 4 };

int fis1\_gRI4[] = { 1, 5 };

int fis1\_gRI5[] = { 2, 1 };

int fis1\_gRI6[] = { 2, 2 };

int fis1\_gRI7[] = { 2, 3 };

int fis1\_gRI8[] = { 2, 4 };

int fis1\_gRI9[] = { 2, 5 };

int fis1\_gRI10[] = { 3, 1 };

int fis1\_gRI11[] = { 3, 2 };

int fis1\_gRI12[] = { 3, 3 };

int fis1\_gRI13[] = { 3, 4 };

int fis1\_gRI14[] = { 3, 5 };

int fis1\_gRI15[] = { 4, 1 };

int fis1\_gRI16[] = { 4, 2 };

int fis1\_gRI17[] = { 4, 3 };

int fis1\_gRI18[] = { 4, 4 };

int fis1\_gRI19[] = { 4, 5 };

int fis1\_gRI20[] = { 5, 1 };

int fis1\_gRI21[] = { 5, 2 };

int fis1\_gRI22[] = { 5, 3 };

int fis1\_gRI23[] = { 5, 4 };

int fis1\_gRI24[] = { 5, 5 };

int\* fis1\_gRI[] = { fis1\_gRI0, fis1\_gRI1, fis1\_gRI2, fis1\_gRI3, fis1\_gRI4, fis1\_gRI5, fis1\_gRI6, fis1\_gRI7, fis1\_gRI8, fis1\_gRI9, fis1\_gRI10, fis1\_gRI11, fis1\_gRI12, fis1\_gRI13, fis1\_gRI14, fis1\_gRI15, fis1\_gRI16, fis1\_gRI17, fis1\_gRI18, fis1\_gRI19, fis1\_gRI20, fis1\_gRI21, fis1\_gRI22, fis1\_gRI23, fis1\_gRI24 };

// Rule Outputs

int fis1\_gRO0[] = { 1 };

int fis1\_gRO1[] = { 2 };

int fis1\_gRO2[] = { 3 };

int fis1\_gRO3[] = { 4 };

int fis1\_gRO4[] = { 5 };

int fis1\_gRO5[] = { 6 };

int fis1\_gRO6[] = { 7 };

int fis1\_gRO7[] = { 8 };

int fis1\_gRO8[] = { 9 };

int fis1\_gRO9[] = { 10 };

int fis1\_gRO10[] = { 11 };

int fis1\_gRO11[] = { 12 };

int fis1\_gRO12[] = { 13 };

int fis1\_gRO13[] = { 14 };

int fis1\_gRO14[] = { 15 };

int fis1\_gRO15[] = { 16 };

int fis1\_gRO16[] = { 17 };

int fis1\_gRO17[] = { 18 };

int fis1\_gRO18[] = { 19 };

int fis1\_gRO19[] = { 20 };

int fis1\_gRO20[] = { 21 };

int fis1\_gRO21[] = { 22 };

int fis1\_gRO22[] = { 23 };

int fis1\_gRO23[] = { 24 };

int fis1\_gRO24[] = { 25 };

int\* fis1\_gRO[] = { fis1\_gRO0, fis1\_gRO1, fis1\_gRO2, fis1\_gRO3, fis1\_gRO4, fis1\_gRO5, fis1\_gRO6, fis1\_gRO7, fis1\_gRO8, fis1\_gRO9, fis1\_gRO10, fis1\_gRO11, fis1\_gRO12, fis1\_gRO13, fis1\_gRO14, fis1\_gRO15, fis1\_gRO16, fis1\_gRO17, fis1\_gRO18, fis1\_gRO19, fis1\_gRO20, fis1\_gRO21, fis1\_gRO22, fis1\_gRO23, fis1\_gRO24 };

// Input range Min

FIS1\_TYPE fis1\_gIMin[] = { 0, 0 };

// Input range Max

FIS1\_TYPE fis1\_gIMax[] = { 5, 15 };

// Output range Min

FIS1\_TYPE fis1\_gOMin[] = { 0 };

// Output range Max

FIS1\_TYPE fis1\_gOMax[] = { 60 };

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Data dependent support functions for Fuzzy Inference System

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// None for Sugeno

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Fuzzy Inference System

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void fis1\_evaluate()

{

   FIS1\_TYPE fuzzy1Input0[] = { 0, 0, 0, 0, 0 };

   FIS1\_TYPE fuzzy1Input1[] = { 0, 0, 0, 0, 0 };

   FIS1\_TYPE\* fuzzy1Input[fis1\_gcI] = { fuzzy1Input0, fuzzy1Input1, };

   FIS1\_TYPE fuzzy1Output0[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

   FIS1\_TYPE\* fuzzy1Output[fis1\_gcO] = { fuzzy1Output0, };

   FIS1\_TYPE fuzzy1Rules[fis1\_gcR] = { 0 };

   FIS1\_TYPE fuzzy1Fires[fis1\_gcR] = { 0 };

   FIS1\_TYPE\* fuzzy1RuleSet[] = { fuzzy1Rules, fuzzy1Fires };

   FIS1\_TYPE sW1 = 0;

   // Transforming input to fuzzy Input

   int i1, j1, r1, o1;

   for (i1 = 0; i1 < fis1\_gcI; ++i1)

   {

       for (j1 = 0; j1 < fis1\_gIMFCount[i1]; ++j1)

       {

           fuzzy1Input[i1][j1] =

               (fis1\_gMF[fis1\_gMFI[i1][j1]])(g\_fis1Input[i1], fis1\_gMFICoeff[i1][j1]);

       }

   }

   int index1 = 0;

   for (r1 = 0; r1 < fis1\_gcR; ++r1)

   {

       if (fis1\_gRType[r1] == 1)

       {

           fuzzy1Fires[r1] = 1;

           for (i1 = 0; i1 < fis1\_gcI; ++i1)

           {

               index1 = fis1\_gRI[r1][i1];

               if (index1 > 0)

                   fuzzy1Fires[r1] = fis1\_prod(fuzzy1Fires[r1], fuzzy1Input[i1][index1 - 1]);

               else if (index1 < 0)

                   fuzzy1Fires[r1] = fis1\_prod(fuzzy1Fires[r1], 1 - fuzzy1Input[i1][-index1 - 1]);

               else

                   fuzzy1Fires[r1] = fis1\_prod(fuzzy1Fires[r1], 1);

           }

       }

       else

       {

           fuzzy1Fires[r1] = 0;

           for (i1 = 0; i1 < fis1\_gcI; ++i1)

           {

               index1 = fis1\_gRI[r1][i1];

               if (index1 > 0)

                   fuzzy1Fires[r1] = fis1\_probor(fuzzy1Fires[r1], fuzzy1Input[i1][index1 - 1]);

               else if (index1 < 0)

                   fuzzy1Fires[r1] = fis1\_probor(fuzzy1Fires[r1], 1 - fuzzy1Input[i1][-index1 - 1]);

               else

                   fuzzy1Fires[r1] = fis1\_probor(fuzzy1Fires[r1], 0);

           }

       }

       fuzzy1Fires[r1] = fis1\_gRWeight[r1] \* fuzzy1Fires[r1];

       sW1 += fuzzy1Fires[r1];

   }

   if (sW1 == 0)

   {

       for (o1 = 0; o1 < fis1\_gcO; ++o1)

       {

           g\_fis1Output[o1] = ((fis1\_gOMax[o1] + fis1\_gOMin[o1]) / 2);

       }

   }

   else

   {

       for (o1 = 0; o1 < fis1\_gcO; ++o1)

       {

           FIS1\_TYPE sWI1 = 0.0;

           for (j1 = 0; j1 < fis1\_gOMFCount[o1]; ++j1)

           {

               fuzzy1Output[o1][j1] = fis1\_gMFOCoeff[o1][j1][fis1\_gcI];

               for (i1 = 0; i1 < fis1\_gcI; ++i1)

               {

                   fuzzy1Output[o1][j1] += g\_fis1Input[i1] \* fis1\_gMFOCoeff[o1][j1][i1];

               }

           }

           for (r1 = 0; r1 < fis1\_gcR; ++r1)

           {

               index1 = fis1\_gRO[r1][o1] - 1;

               sWI1 += fuzzy1Fires[r1] \* fuzzy1Output[o1][index1];

           }

           g\_fis1Output[o1] = sWI1 / sW1;

       }

   }

}

void fis2\_evaluate()

{

   FIS1\_TYPE fuzzy1Input0[] = { 0, 0, 0, 0, 0 };

   FIS1\_TYPE fuzzy1Input1[] = { 0, 0, 0, 0, 0 };

   FIS1\_TYPE\* fuzzy1Input[fis1\_gcI] = { fuzzy1Input0, fuzzy1Input1, };

   FIS1\_TYPE fuzzy1Output0[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

   FIS1\_TYPE\* fuzzy1Output[fis1\_gcO] = { fuzzy1Output0, };

   FIS1\_TYPE fuzzy1Rules[fis1\_gcR] = { 0 };

   FIS1\_TYPE fuzzy1Fires[fis1\_gcR] = { 0 };

   FIS1\_TYPE\* fuzzy1RuleSet[] = { fuzzy1Rules, fuzzy1Fires };

   FIS1\_TYPE sW1 = 0;

   // Transforming input to fuzzy Input

   int i1, j1, r1, o1;

   for (i1 = 0; i1 < fis1\_gcI; ++i1)

   {

       for (j1 = 0; j1 < fis1\_gIMFCount[i1]; ++j1)

       {

           fuzzy1Input[i1][j1] =

               (fis1\_gMF[fis1\_gMFI[i1][j1]])(g\_fis2Input[i1], fis1\_gMFICoeff[i1][j1]);

       }

   }

   int index1 = 0;

   for (r1 = 0; r1 < fis1\_gcR; ++r1)

   {

       if (fis1\_gRType[r1] == 1)

       {

           fuzzy1Fires[r1] = 1;

           for (i1 = 0; i1 < fis1\_gcI; ++i1)

           {

               index1 = fis1\_gRI[r1][i1];

               if (index1 > 0)

                   fuzzy1Fires[r1] = fis1\_prod(fuzzy1Fires[r1], fuzzy1Input[i1][index1 - 1]);

               else if (index1 < 0)

                   fuzzy1Fires[r1] = fis1\_prod(fuzzy1Fires[r1], 1 - fuzzy1Input[i1][-index1 - 1]);

               else

                   fuzzy1Fires[r1] = fis1\_prod(fuzzy1Fires[r1], 1);

           }

       }

       else

       {

           fuzzy1Fires[r1] = 0;

           for (i1 = 0; i1 < fis1\_gcI; ++i1)

           {

               index1 = fis1\_gRI[r1][i1];

               if (index1 > 0)

                   fuzzy1Fires[r1] = fis1\_probor(fuzzy1Fires[r1], fuzzy1Input[i1][index1 - 1]);

               else if (index1 < 0)

                   fuzzy1Fires[r1] = fis1\_probor(fuzzy1Fires[r1], 1 - fuzzy1Input[i1][-index1 - 1]);

               else

                   fuzzy1Fires[r1] = fis1\_probor(fuzzy1Fires[r1], 0);

           }

       }

       fuzzy1Fires[r1] = fis1\_gRWeight[r1] \* fuzzy1Fires[r1];

       sW1 += fuzzy1Fires[r1];

   }

   if (sW1 == 0)

   {

       for (o1 = 0; o1 < fis1\_gcO; ++o1)

       {

           g\_fis2Output[o1] = ((fis1\_gOMax[o1] + fis1\_gOMin[o1]) / 2);

       }

   }

   else

   {

       for (o1 = 0; o1 < fis1\_gcO; ++o1)

       {

           FIS1\_TYPE sWI1 = 0.0;

           for (j1 = 0; j1 < fis1\_gOMFCount[o1]; ++j1)

           {

               fuzzy1Output[o1][j1] = fis1\_gMFOCoeff[o1][j1][fis1\_gcI];

               for (i1 = 0; i1 < fis1\_gcI; ++i1)

               {

                   fuzzy1Output[o1][j1] += g\_fis2Input[i1] \* fis1\_gMFOCoeff[o1][j1][i1];

               }

           }

           for (r1 = 0; r1 < fis1\_gcR; ++r1)

           {

               index1 = fis1\_gRO[r1][o1] - 1;

               sWI1 += fuzzy1Fires[r1] \* fuzzy1Output[o1][index1];

           }

           g\_fis2Output[o1] = sWI1 / sW1;

       }

   }

}

void fis3\_evaluate()

{

   FIS1\_TYPE fuzzy1Input0[] = { 0, 0, 0, 0, 0 };

   FIS1\_TYPE fuzzy1Input1[] = { 0, 0, 0, 0, 0 };

   FIS1\_TYPE\* fuzzy1Input[fis1\_gcI] = { fuzzy1Input0, fuzzy1Input1, };

   FIS1\_TYPE fuzzy1Output0[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

   FIS1\_TYPE\* fuzzy1Output[fis1\_gcO] = { fuzzy1Output0, };

   FIS1\_TYPE fuzzy1Rules[fis1\_gcR] = { 0 };

   FIS1\_TYPE fuzzy1Fires[fis1\_gcR] = { 0 };

   FIS1\_TYPE\* fuzzy1RuleSet[] = { fuzzy1Rules, fuzzy1Fires };

   FIS1\_TYPE sW1 = 0;

   // Transforming input to fuzzy Input

   int i1, j1, r1, o1;

   for (i1 = 0; i1 < fis1\_gcI; ++i1)

   {

       for (j1 = 0; j1 < fis1\_gIMFCount[i1]; ++j1)

       {

           fuzzy1Input[i1][j1] =

               (fis1\_gMF[fis1\_gMFI[i1][j1]])(g\_fis3Input[i1], fis1\_gMFICoeff[i1][j1]);

       }

   }

   int index1 = 0;

   for (r1 = 0; r1 < fis1\_gcR; ++r1)

   {

       if (fis1\_gRType[r1] == 1)

       {

           fuzzy1Fires[r1] = 1;

           for (i1 = 0; i1 < fis1\_gcI; ++i1)

           {

               index1 = fis1\_gRI[r1][i1];

               if (index1 > 0)

                   fuzzy1Fires[r1] = fis1\_prod(fuzzy1Fires[r1], fuzzy1Input[i1][index1 - 1]);

               else if (index1 < 0)

                   fuzzy1Fires[r1] = fis1\_prod(fuzzy1Fires[r1], 1 - fuzzy1Input[i1][-index1 - 1]);

               else

                   fuzzy1Fires[r1] = fis1\_prod(fuzzy1Fires[r1], 1);

           }

       }

       else

       {

           fuzzy1Fires[r1] = 0;

           for (i1 = 0; i1 < fis1\_gcI; ++i1)

           {

               index1 = fis1\_gRI[r1][i1];

               if (index1 > 0)

                   fuzzy1Fires[r1] = fis1\_probor(fuzzy1Fires[r1], fuzzy1Input[i1][index1 - 1]);

               else if (index1 < 0)

                   fuzzy1Fires[r1] = fis1\_probor(fuzzy1Fires[r1], 1 - fuzzy1Input[i1][-index1 - 1]);

               else

                   fuzzy1Fires[r1] = fis1\_probor(fuzzy1Fires[r1], 0);

           }

       }

       fuzzy1Fires[r1] = fis1\_gRWeight[r1] \* fuzzy1Fires[r1];

       sW1 += fuzzy1Fires[r1];

   }

   if (sW1 == 0)

   {

       for (o1 = 0; o1 < fis1\_gcO; ++o1)

       {

           g\_fis3Output[o1] = ((fis1\_gOMax[o1] + fis1\_gOMin[o1]) / 2);

       }

   }

   else

   {

       for (o1 = 0; o1 < fis1\_gcO; ++o1)

       {

           FIS1\_TYPE sWI1 = 0.0;

           for (j1 = 0; j1 < fis1\_gOMFCount[o1]; ++j1)

           {

               fuzzy1Output[o1][j1] = fis1\_gMFOCoeff[o1][j1][fis1\_gcI];

               for (i1 = 0; i1 < fis1\_gcI; ++i1)

               {

                   fuzzy1Output[o1][j1] += g\_fis3Input[i1] \* fis1\_gMFOCoeff[o1][j1][i1];

               }

           }

           for (r1 = 0; r1 < fis1\_gcR; ++r1)

           {

               index1 = fis1\_gRO[r1][o1] - 1;

               sWI1 += fuzzy1Fires[r1] \* fuzzy1Output[o1][index1];

           }

           g\_fis3Output[o1] = sWI1 / sW1;

       }

   }

}

void fis4\_evaluate()

{

   FIS1\_TYPE fuzzy1Input0[] = { 0, 0, 0, 0, 0 };

   FIS1\_TYPE fuzzy1Input1[] = { 0, 0, 0, 0, 0 };

   FIS1\_TYPE\* fuzzy1Input[fis1\_gcI] = { fuzzy1Input0, fuzzy1Input1, };

   FIS1\_TYPE fuzzy1Output0[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

   FIS1\_TYPE\* fuzzy1Output[fis1\_gcO] = { fuzzy1Output0, };

   FIS1\_TYPE fuzzy1Rules[fis1\_gcR] = { 0 };

   FIS1\_TYPE fuzzy1Fires[fis1\_gcR] = { 0 };

   FIS1\_TYPE\* fuzzy1RuleSet[] = { fuzzy1Rules, fuzzy1Fires };

   FIS1\_TYPE sW1 = 0;

   // Transforming input to fuzzy Input

   int i1, j1, r1, o1;

   for (i1 = 0; i1 < fis1\_gcI; ++i1)

   {

       for (j1 = 0; j1 < fis1\_gIMFCount[i1]; ++j1)

       {

           fuzzy1Input[i1][j1] =

               (fis1\_gMF[fis1\_gMFI[i1][j1]])(g\_fis4Input[i1], fis1\_gMFICoeff[i1][j1]);

       }

   }

   int index1 = 0;

   for (r1 = 0; r1 < fis1\_gcR; ++r1)

   {

       if (fis1\_gRType[r1] == 1)

       {

           fuzzy1Fires[r1] = 1;

           for (i1 = 0; i1 < fis1\_gcI; ++i1)

           {

               index1 = fis1\_gRI[r1][i1];

               if (index1 > 0)

                   fuzzy1Fires[r1] = fis1\_prod(fuzzy1Fires[r1], fuzzy1Input[i1][index1 - 1]);

               else if (index1 < 0)

                   fuzzy1Fires[r1] = fis1\_prod(fuzzy1Fires[r1], 1 - fuzzy1Input[i1][-index1 - 1]);

               else

                   fuzzy1Fires[r1] = fis1\_prod(fuzzy1Fires[r1], 1);

           }

       }

       else

       {

           fuzzy1Fires[r1] = 0;

           for (i1 = 0; i1 < fis1\_gcI; ++i1)

           {

               index1 = fis1\_gRI[r1][i1];

               if (index1 > 0)

                   fuzzy1Fires[r1] = fis1\_probor(fuzzy1Fires[r1], fuzzy1Input[i1][index1 - 1]);

               else if (index1 < 0)

                   fuzzy1Fires[r1] = fis1\_probor(fuzzy1Fires[r1], 1 - fuzzy1Input[i1][-index1 - 1]);

               else

                   fuzzy1Fires[r1] = fis1\_probor(fuzzy1Fires[r1], 0);

           }

       }

       fuzzy1Fires[r1] = fis1\_gRWeight[r1] \* fuzzy1Fires[r1];

       sW1 += fuzzy1Fires[r1];

   }

   if (sW1 == 0)

   {

       for (o1 = 0; o1 < fis1\_gcO; ++o1)

       {

           g\_fis4Output[o1] = ((fis1\_gOMax[o1] + fis1\_gOMin[o1]) / 2);

       }

   }

   else

   {

       for (o1 = 0; o1 < fis1\_gcO; ++o1)

       {

           FIS1\_TYPE sWI1 = 0.0;

           for (j1 = 0; j1 < fis1\_gOMFCount[o1]; ++j1)

           {

               fuzzy1Output[o1][j1] = fis1\_gMFOCoeff[o1][j1][fis1\_gcI];

               for (i1 = 0; i1 < fis1\_gcI; ++i1)

               {

                   fuzzy1Output[o1][j1] += g\_fis4Input[i1] \* fis1\_gMFOCoeff[o1][j1][i1];

               }

           }

           for (r1 = 0; r1 < fis1\_gcR; ++r1)

           {

               index1 = fis1\_gRO[r1][o1] - 1;

               sWI1 += fuzzy1Fires[r1] \* fuzzy1Output[o1][index1];

           }

           g\_fis4Output[o1] = sWI1 / sW1;

       }

   }

}

void loop() {

**Serial**.println("Checking...");

checkSensor1();

checkSensor2();

checkSensor3();

checkSensor4();

//Serial.println("Done checking.");

// Serial.print(lane1);

// Serial.print(lane2);

// Serial.print(lane3);

// Serial.print(lane4);

// Serial.print("\n");

int countA = lane1;

int countB = lane2;

int countC = lane3;

int countD = lane4;

checkSensor1();

checkSensor2();

checkSensor3();

checkSensor4();

//Enter input to Fuzzy Logic

   g\_fisInput[0] = countA;

   g\_fisInput[1] = countB;

   g\_fisInput[2] = countC;

   g\_fisInput[3] = countD;

checkSensor1();

checkSensor2();

checkSensor3();

checkSensor4();

   //Enter input to ANFIS

   g\_fis1Input[0]= countA;

**Serial**.print("Count Lane A: ");

**Serial**.println(g\_fis1Input[0]);

   g\_fis1Input[1]= countB+countC+countD;

   g\_fis2Input[0]= countB;

**Serial**.print("Count Lane B: ");

**Serial**.println(g\_fis2Input[0]);

   g\_fis2Input[1]= countA+countC+countD;

   g\_fis3Input[0]= countC;

**Serial**.print("Count Lane C: ");

**Serial**.println(g\_fis3Input[0]);

   g\_fis3Input[1]= countA+countB+countD;

   g\_fis4Input[0]=countD;

**Serial**.print("Count Lane D: ");

**Serial**.println(g\_fis4Input[0]);

   g\_fis4Input[1]= countA+countB+countC;

   fis\_evaluate();

   fis1\_evaluate();

   fis2\_evaluate();

   fis3\_evaluate();

   fis4\_evaluate();

**Serial**.print("Sequence: ");

   float sequence = round(g\_fisOutput[0]);

**Serial**.println(sequence);

   //Duration Green Light

   //Lane A

   float greenA = (g\_fis1Output[0])\*100;

**Serial**.print("Green Light Duration (s) Line A: ");

**Serial**.println(greenA/100);

   //Lane B

   float greenB = (g\_fis2Output[0])\*100;

**Serial**.print("Green Light Duration (s) Line B: ");

**Serial**.println(greenB/100);

   //Lane C

   float greenC =(g\_fis3Output[0])\*100;

**Serial**.print("Green Light Duration (s) Line C: ");

**Serial**.println(greenC/100);

   //Lane D

   float greenD = (g\_fis4Output[0])\*100;

**Serial**.print("Green Light Duration (s) Line D: ");

**Serial**.println(greenD/100);

   //Let all red LED turn on at first

   digitalWrite(rA, HIGH);

   digitalWrite(rB, HIGH);

   digitalWrite(rC, HIGH);

   digitalWrite(rD, HIGH);

   if(sequence == 1)

   {

**Serial**.println("ABCD");

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     lane2=0;

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

   }

   if(sequence == 2)

   {

**Serial**.println("ABDC");

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

   }

   if(sequence == 3)

   {

**Serial**.println("ACBD");

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

      //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

   }

   if(sequence == 4)

   {

**Serial**.println("ACDB");

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

      //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

   }

   if(sequence == 5)

   {

**Serial**.println("ADBC");

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

   }

   if(sequence == 6)

   {

**Serial**.println("ADCB");

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

   }

   if(sequence == 7)

   {

**Serial**.println("BACD");

       //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

   }

   if(sequence == 8)

   {

**Serial**.println("BADC");

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

   }

   if(sequence == 9)

   {

**Serial**.println("BACD");

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

      //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

   }

   if(sequence == 10)

   {

**Serial**.println("BCDA");

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

      //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

   }

   if(sequence == 11)

   {

**Serial**.println("BDAC");

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

   }

   if(sequence == 12)

   {

**Serial**.println("BDCA");

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

   }

   if(sequence == 13)

   {

**Serial**.println("CABD");

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

   }

   if(sequence == 14)

   {

**Serial**.println("CADB");

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

   }

   if(sequence == 15)

   {

**Serial**.println("CBAD");

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

   }

   if(sequence == 16)

   {

**Serial**.println("CBDA");

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

   }

   if(sequence == 17)

   {

**Serial**.println("CDAB");

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

   }

   if(sequence == 18)

   {

**Serial**.println("CDBA");

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

   }

   if(sequence == 19)

   {

**Serial**.println("DABC");

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

   }

   if(sequence == 20)

   {

**Serial**.println("DACB");

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

   }

   if(sequence == 21)

   {

**Serial**.println("DBAC");

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

   }

   if(sequence == 22)

   {

**Serial**.println("DBCA");

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

      //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

   }

   if(sequence == 23)

   {

**Serial**.println("DCAB");

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

      //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

      //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

   }

   if(sequence == 24)

   {

**Serial**.println("DCBA");

     //Lane D

     digitalWrite(rD, LOW);

     digitalWrite(gD, HIGH);

     delay(greenD);

     digitalWrite(gD, LOW);

     digitalWrite(yD, HIGH);

     delay(1500);

     digitalWrite(yD, LOW);

     digitalWrite(rD, HIGH);

     //Lane C

     digitalWrite(rC, LOW);

     digitalWrite(gC, HIGH);

     delay(greenC);

     digitalWrite(gC, LOW);

     digitalWrite(yC, HIGH);

     delay(1500);

     digitalWrite(yC, LOW);

     digitalWrite(rC, HIGH);

     //Lane B

     digitalWrite(rB, LOW);

     digitalWrite(gB, HIGH);

     delay(greenB);

     digitalWrite(gB, LOW);

     digitalWrite(yB, HIGH);

     delay(1500);

     digitalWrite(yB, LOW);

     digitalWrite(rB, HIGH);

      //Lane A

     digitalWrite(rA, LOW);

     digitalWrite(gA, HIGH);

     delay(greenA);

     digitalWrite(gA, LOW);

     digitalWrite(yA, HIGH);

     delay(1500);

     digitalWrite(yA, LOW);

     digitalWrite(rA, HIGH);

   }

   else

   {

     digitalWrite(rA, HIGH);

     digitalWrite(rB, HIGH);

     digitalWrite(rC, HIGH);

     digitalWrite(rD, HIGH);

   }

checkSensor1();

checkSensor2();

checkSensor3();

checkSensor4();

}