//there are about 25 functions in this robot control script

//

//give power to robot platform, then connect the data wire with a computer

//wire mode (could be control through wireless tech., Wi-Fi)

//select and start (single servo control/ group action)

//========================================================================================================================================================================

// ----------LIBRARIES--------------

#include "VL53L0X2.h"; //the bookshelf we create

#include "Adafruit\_VL53L0X2.h"

#include "LobotServoController.h"

#include "Ultrasound.h"

//#include "Screen.h"

#include <SoftwareSerial.h>

#include <Arduino.h>//default

#include <Wire.h>

//byte 8 bit

//short 16 bit

//int 32 bit

//long 64 bit

//there is no unsigned byte in Arduino //0 - 255 //A byte stores an 8-bit unsigned number, from 0 to 255.

//unsigned short //0 - 65535

//unsigned int //0 - (2^32 - 1)

//unsigned long

// --------CONSTANTS (won't change)-------------------------------------------------------

// address we will assign if dual sensor is present

#define LOX1\_ADDRESS 0x30//0x27 and 0x28 could be used

#define LOX2\_ADDRESS 0x29//default

#define LOX3\_ADDRESS 0x31

#define LOX4\_ADDRESS 0x32

// set the pins to shutdown

#define SHT\_LOX1 6//sensor1//Servo7PositionSensor

#define SHT\_LOX2 A0//sensor2//Servo7PositionSensor

#define SHT\_LOX3 7//sensor3

#define SHT\_LOX4 A1//sensor4

const byte IR1 = 2; //infrared sensor

//const byte IR1 = 3; //infrared sensor

const byte IR2 = 8; //infrared sensor

//const byte IR2 = 9; //infrared sensor

const byte IR3 = 10; //infrared sensor

//const byte IR3 = 11; //infrared sensor

const byte RxPin = 4; //robot arm port

const byte TxPin = 5; //robot arm port

const byte LED = 13;

const byte buttonPin = 12; //Touch sensor

//from the reference point 265 to the 10,000 ms point

//distance sensor value from 265 to about 350

// (350 - 265)/ 10,000 = 0.0084/ ms = 8.4/s

float speedTrain = 0.0084;

//assume the sensor1 value will not stuck in a constant value when the robot arm moves

//assumw the pet will not stand between the sensor and the robot arm

//move train only if the sensor value is from sensor1 0 to 710

//protection:

//moveForward(durationMoveTrain\_Max); allowed max distance //moveForward(durationMoveTrain\_referPoint2jointPoint); in the setup() which means sensor1 310 joint point

unsigned int durationMoveTrain\_OriginalReferPoint2jointPoint = 15000;

unsigned int durationMoveTrain\_referPoint2jointPoint = durationMoveTrain\_OriginalReferPoint2jointPoint;

unsigned short jointPoint = 310; //before the point, the sensor value is accurate believe.

//sensor1 sensor1\_Min allowed min distance

unsigned short sensor1\_Min = 110;

//reference point: 145 (143 - 147 is fine)

unsigned short referencePoint = 145; //default// should > 2 //should > sensor1\_Min

//trainDuration = (147 - 143)/ speedTrain = 595.24 ms

unsigned short trainDuration = 595; //19; //952; //1000; //define the smallest motor step distance//resolution;

float safeIndex = 0.8; // 0 < safeIndex < 1

//The sensor1 wall (mirror) is 6.5 cm far from the edge of platform.

//The barrier, say pet, cannot be under the edge

//80 is the ideal backward sensor1 range (after the 308 point) per 10 seconds

//10.3 is the ideal backward distance (after the 308 point) per 10 seconds

//80/ 10.3 = x/ 6.5

//x = 80 \* 6.5/ 10.3 = 50.4854

//Therefore, the allowed distance variance is no more than 50.4854

//max distance Changed per trainDuration// arrange = 50.4854/ (trainDuration \* speedTrain) = 10.1011;

//it is better for us to choose the index less than or equal to 10

byte barrierIndex = 9;

//could be initialized again\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

unsigned int durationMoveTrain\_Max = durationMoveTrain\_referPoint2jointPoint + 30000; //45000;

//dictionary:

//the trashbin durationMoveTrain\_Max, table 25000, the duration of train movement

unsigned int durationMoveTrain\_trashBin = durationMoveTrain\_Max;

unsigned int durationMoveTrain\_table = durationMoveTrain\_referPoint2jointPoint + 10000;

//------------ VARIABLES (will change)-----------------------------------------------------

unsigned long currentMillis = 0; //stores the value of total run time

//loopPeriod = duration + interval;

//period = interval + duration

//if switch between two states is needed, then setting clock below:

//if (state1) {

// set clock by using interval

// } else {//state2

// set clock by using duration

// }

//in this class only duration or interval period is used

// objects for the vl53l0x

Adafruit\_VL53L0X2 lox1; //VL53L0X2 lox1;

//VL53L0X2 lox2;

VL53L0X2 lox3;

VL53L0X2 lox4;

SoftwareSerial mySerial(RxPin, TxPin); //Instantiate soft serial port（rx，tx）

LobotServoController robotController(mySerial); //Instantiate servo controller communication library

Ultrasound ultrasound; //Instantiate the ultrasonic class

//Screen screen;

/\*

//why it cause dead loop if we call constructors in setup()

lox1 = Adafruit\_VL53L0X2(); //lox1 = VL53L0X2();

//lox2 = VL53L0X2();

lox3 = VL53L0X2();

lox4 = VL53L0X2();

//robotController(mySerial);

ultrasound = Ultrasound(); //Instantiate the ultrasonic class

//screen = Screen();

\*/

bool isRunTime = false;

//could be initialized again\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//assume servo7 is controlled by Arduino board, Hiwonder robot arm board support it by providing energy and delivering information.

bool isRunningTrain = false;

//There are two methods to control the position of the train, servo7:

//because the sensor wire is poor, two different methods are used to control the position of the train, servo7:

//when train is too far from the sensor

//keeps or locks the previous time point/value/variable when visited joint point,

//the time point as the reference when robot moves forward and farther away from the sensor.

//1. Time setting (after/on the right side of the joint point)

//////records the previous time point only (once) at the beginning of train movement, like this:

//////////if (allow2updatePreviousMoveServo7\_timer == true) {}

//////keeps or locks the previous time point/value/variable during the whole train movement process:

//////////allow2updatePreviousMoveServo7\_timer = false;

//////////previousMoveServo7\_timer = currentMillis;

unsigned long previousMoveServo7\_timer = 0; //it is not enough to use int // stores the value of currentMillis as the previous time point for control the time of servo7 movement

//reset the previousMoveServo7\_timer automatically ONLY IF the train will begin to move.

//private // this variable is not allowed to be edited by users

//There are two situations in which the train will begin to move:

// (a) when users ask to move train or

// (b) when the train is not hindered and it continues to finish the previous movement.

bool allow2updatePreviousMoveServo7\_timer = false; //could be true, ONLY IF the train will begin to move.

//besides, we can use the same variable, durationMoveTrain, to represent two physical rules:

//1. Represent the total time point when there is no barrier. In other words, planned duration of the train Movement.

//////////durationMoveTrain = targeted duration; //For example, durationMoveTrain = 5000; //meaning set 5s movement

//////////allow2updatePreviousMoveServo7\_timer = true; //unlock

//2. Represent the remainning time period when there is a barrier.

//////////durationMoveTrain = abs3(durationMoveTrain\_remainingPlanned - trainDuration); //abs3() only applied to the integer

//durationMoveTrain\_accumulationActual = abs2(currentMillis - previousMoveServo7\_timer);

//////////allow2updatePreviousMoveServo7\_timer = true; //unlock

//2. Distance setting (before/on the left side of the joint point)

//when train is close to the sensor, it is fine, just use the sensor value

// through previousDistanceSensor1, by recording sensor1 value and comparing previous and current value,

//we can detect the movement accuracy or whether the sensor is hindered.

//integer distance;

unsigned short previousDistanceSensor1 = 0;

unsigned short sensor1 = 0; //train

//previousDistanceSensor1 = sensor1; //update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

unsigned int durationMoveTrain = 0;

//besides, we can use the same variable, durationMoveTrain, to represent two physical rules:

//1. Represent the total time point when there is no barrier. In other words, planned duration of the train Movement.

//////////durationMoveTrain = targeted duration; //For example, durationMoveTrain = 5000; //meaning set 5s movement

//////////allow2updatePreviousMoveServo7\_timer = true; //unlock

//2. Represent the remainning time period when there is a barrier.

//////////durationMoveTrain = abs3(durationMoveTrain\_remainingPlanned - trainDuration); //abs3() only applied to the integer

//durationMoveTrain\_accumulationActual = abs2(currentMillis - previousMoveServo7\_timer);

//////////allow2updatePreviousMoveServo7\_timer = true; //unlock

unsigned int durationMoveTrain\_remainingPlanned;

bool isFirstTime2SetDurationMoveTrain\_remainingPlanned = true;

//durationMoveTrain\_remainingPlanned = durationMoveTrain; // the first time

//since the second time,

//durationMoveTrain\_remainingPlanned = abs3(durationMoveTrain\_remainingPlanned(previousDurationMoveTrain) - trainDuration); //abs3() can only be used for integer

//only (once) at the beginning of train movement (before recover from the hindered state),

//the durationMoveTrain\_remainingPlanned will be recorded by durationMoveTrain.

//1.only if the train finish the required movement or

//2.at the begining of train movement (user set rather than recover from the hindered state),

//this variable will be updated/reset.

unsigned long previousMillis;

//unsigned short sensor2 = 0;

unsigned short sensor3 = 0;

unsigned short sensor4 = 0;

unsigned short distance = 0; //ultrasound sensor

//Record the number of touches

byte NumTouch;

//byte buttonLed\_State = LOW;

byte isHereArray = 0x00;

byte isSafe2grib = 0;

//before power off, it is very important

//put the good on the ground

//before nap or stuck in some dead loop

//pauseTrain();

//because it is not a right stop

//besides, if isRunning Train = 0, group action will continue to be run wrongly.

//robotController.stopActionGroup();

//========================================================================================================================================================================

void setup() {

//no need to be initialized again

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//initial master boards' interface

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Serial.begin(9600);

// wait until serial port opens for native USB devices

while (! Serial) { delay(1); }

mySerial.begin(9600); //communication between Uno R3 kit and the robot arm

//Set the functions of IO port for each infrared sensor

pinMode(IR1, INPUT);

pinMode(IR2, INPUT);

pinMode(IR3, INPUT);

pinMode(LED, OUTPUT);

pinMode(buttonPin, INPUT);

digitalWrite(LED, LOW);

// set the button pin as input with a pullup resistor to ensure it defaults to HIGH

pinMode(buttonPin, INPUT\_PULLUP);

pinMode(SHT\_LOX1, OUTPUT);

pinMode(SHT\_LOX2, OUTPUT);

pinMode(SHT\_LOX3, OUTPUT);

pinMode(SHT\_LOX4, OUTPUT);

//Serial.println(F("Shutdown TOF sensor pins inited..."));

digitalWrite(SHT\_LOX1, LOW);

digitalWrite(SHT\_LOX2, LOW);

digitalWrite(SHT\_LOX3, LOW);

digitalWrite(SHT\_LOX4, LOW);

//Serial.println(F("Both TOF sensors in reset mode...(pins are low)"));

//Serial.println(F("Starting..."));

//initial sensors

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

setID();

delay(100); // the TOF distance sensors need 90 ms to read

initialization(); //read\_multi\_tof\_sensors(); //previousDistanceSensor1 = sensor1;

//loopInSetup();

//Sheridan framework

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

{

//deal with the corner case

delay(100); // the TOF distance sensors need 90 ms to read

read\_multi\_tof\_sensors();

//because sometimes train stop without recording the correct sensor1 value into previousDistanceSensor1

previousDistanceSensor1 = sensor1;

try2Recover();

delay(100); // the TOF distance sensors need 90 ms to read

read\_multi\_tof\_sensors();

previousDistanceSensor1 = sensor1; //assume the sensor1 laser beam is not hindered at the very beginnings

//but it is a corner case

loopInSetup();

//test or some preaction

//delay(100); // the TOF distance sensors need 90 ms to read

//read\_multi\_tof\_sensors();

//previousDistanceSensor1 = sensor1; //assume the sensor1 laser beam is not hindered at the very beginnings

//but it is a corner case

//isRunTime = false;

//moveForward(durationMoveTrain\_table); //moveForward(durationMoveTrain\_referPoint2jointPoint); in the setup() which means sensor1 310 joint point

//delay(100); // the TOF distance sensors need 90 ms to read

//read\_multi\_tof\_sensors();

//previousDistanceSensor1 = sensor1; //assume the sensor1 laser beam is not hindered at the very beginnings

//but it is a corner case

//isRunTime = false;

//moveBackward(10000);

delay(100); // the TOF distance sensors need 90 ms to read

read\_multi\_tof\_sensors();

previousDistanceSensor1 = sensor1; //assume the sensor1 laser beam is not hindered at the very beginnings

//but it is a corner case

}

//Serial.println(F("Runtime"));

isRunTime = true;

}

//========================================================================================================================================================================

void loop() {

// Notice that none of the action happens in loop() {} apart from reading millis()

// it just calls the functions that have the action code

//do not use while() loop or delay() in the loop() {}

//use synchronized timer/counter or asynchronous timer/counter

//set same clock or different clocks

//initial

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//initialization();

currentMillis = millis();

// capture the latest value of millis()

// this is equivalent to noting the time from a clock

// use the same time for all operators to keep them synchronized

// in order to synchronize operators in each iteration of loop() {}, such as robot arm (servro1-6), a train (servo7)

isHereArray = 0x00;

//read

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//user IO

readButton(); //asynchronous, fast

//sensors

if (isHere(IR3))//the farthest InfraRedSensor

{

isHereArray = isHereArray | 0x01; // isHereArray = 0x00(0000 0000) | 0x01(0000 0001);

}

if (isHere(IR2))//the middle InfraRedSensor

{

isHereArray = isHereArray | 0x02; // isHereArray = 0x00(0000 0000) | 0x02(0000 0010);

}

if (isHere(IR1))//the closest InfraRedSensor

{

isHereArray = isHereArray | 0x04; // isHereArray = 0x00(0000 0000) | 0x04(0000 0100);

}

if (isHere())//altrasound

{

isHereArray = isHereArray | 0x08; // isHereArray = 0x00(0000 0000) | 0x08(0000 1000);

}

//asynchronous, fast

//Serial.println("isHereInfraRedSensor(IR1)");

//Serial.println(isHere(IR1));

//Serial.println(IR2);

//digitalRead() could not be put into the timer/counter {}

//Serial.println(isHere());

//isHereArray recording the position of object

//essential sensors to control operators

read\_multi\_tof\_sensors(); //asynchronous, fast

readRobot();

//Receive data returned by the robotic arm

//asynchronous, fast

//analysis

//make decisions

//take actions //synchronize, normal

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

run();

//Serial.println(F("isSafe2grib"));

//Serial.println(isSafe2grib);

bool isDoneStep1 = step1();

if (isDoneStep1) {

step3();

}

//display //report to the PC or monitor

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//screen.draw(sensor3); //asynchronous, very slow

//delay(100); // do not use delay() in the loop() {}, it will cause the currentMillis delay 100 ms, which could undermine the servo7 movement 100 \* 0.0105 = about 1 cm

}

//====================================================================================================================================================================

void initialization() {

//could be initialized again\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

durationMoveTrain\_Max = durationMoveTrain\_referPoint2jointPoint + 30000; //45000;

//dictionary:

//the trashbin durationMoveTrain\_Max, table 25000, the duration of train movement

durationMoveTrain\_trashBin = durationMoveTrain\_Max;

durationMoveTrain\_table = durationMoveTrain\_referPoint2jointPoint + 10000;

//could be initialized again\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//assume servo7 is controlled by arduino board, Hiwonder robot arm board support it by providing energy and delivering information.

isRunningTrain = false;

previousMoveServo7\_timer = 0; //it is not enough to use int // stores the value of currentMillis as the previous time point for control the time of servo7 movement

allow2updatePreviousMoveServo7\_timer = false;

//integer distance;

previousDistanceSensor1 = 0;

sensor1 = 0; //train

//previousDistanceSensor1 = sensor1; //update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

durationMoveTrain = 0;

durationMoveTrain\_remainingPlanned = 0;

isFirstTime2SetDurationMoveTrain\_remainingPlanned = true;

previousMillis = 0;

//unsigned short sensor2 = 0;

sensor3 = 0;

sensor4 = 0;

distance = 0; //ultrasound sensor

//Record the number of touches

NumTouch = 0;

//byte buttonLed\_State = LOW;

//the closest InfraRedSensor and altrasound sensor array

//reflect the object position

isHereArray = 0x00;

isSafe2grib = 0;

//could be initialized again\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

read\_multi\_tof\_sensors();

previousDistanceSensor1 = sensor1; //assume the sensor1 laser beam is not hindered at the very beginnings

//but it is a corner case

//initial timers/counters

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

currentMillis = millis();

// capture the latest value of millis()

// this is equivalent to noting the time from a clock

// use the same time for all operators to keep them synchronized

// in order to synchronize operators in each iteration of loop() {}, such as robot arm (servro1-6), a train (servo7)

//make sure the currentMillis > MAX(trainDuration, MAX(durationMoveTrain))

currentMillis = millis();

if (currentMillis <= durationMoveTrain\_Max) {

unsigned int temp = durationMoveTrain\_Max - currentMillis + 100; //unsigned int ////0 - (2^32 - 1)

//Serial.print(F("Initializing system for "));

//Serial.print(temp/1000);

//Serial.println(F(" seconds"));

//delay(temp);

currentMillis = millis();

} else {//currentMillis > durationMoveTrain\_Max

//skip

}

}

//Run button function

//========================================================================================================================================================================

void run() {

//analysis

//make decisions

//take actions //synchronize, normal

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//initial

static byte step = 0;

//Serial.println(F("step"));

//Serial.println(step);

static unsigned long previousMillis\_timer; //set loop check clock //unsigned long

if(previousMillis\_timer <= abs2(currentMillis - (long)trainDuration))//synchronize

{

previousMillis\_timer += trainDuration; //set clock

//all senstences below follow the same clock

//because we do not need to set different clocks

{

if ((NumTouch >= 1 && NumTouch <= 6) || NumTouch == 10) {//NumTouch = 1 - 4, 5, 6, or 10

if (NumTouch == 5) {//initial

//skip

//do nothing

} else {// (NumTouch != 5) //1s - 3s,15s, or wait

if (NumTouch == 4) {//wait

step = 0; //step = 0;

//wait here until the NumTouch is changed again

//Serial.println(F("wait here until the NumTouch is changed again, button4"));

} else {//NumTouch == 1, 2, 3, 6 or 10

if (NumTouch == 10) {//wait, besides, it means the robot is in front of the table right now

step = 0; //step = 0;

//wait here until the NumTouch is changed again

//Serial.println(F("wait here until the NumTouch is changed again, button10"));

} else {//NumTouch == 1, 2, 3, or 6

//1s - 2s forward, or 1s backward, or 15s forward

if (step == 1) {

if (NumTouch == 3) {//backward

//currentMillis = millis();

moveBackwardSafe();

} else {//forward

//currentMillis = millis();

moveForwardSafe();

}

if (!isRunningTrain) {

step = 2;

}

} else {

if (step == 2) {

if (NumTouch == 6) {//jump to the NumTouch 10, meaning the robot is in front of table

isSafe2grib = 1;

NumTouch = 10;

} else {

NumTouch = 4; //jump out to skip

}

} else {//default://step == 0:

if (NumTouch == 6) {

//Override the value of durationMoveTrain

durationMoveTrain = durationMoveTrain\_table; //durationMoveTrain = 15000; //durationMoveTrain = NumTouch \* 1000; //durationMoveTrain = durationMoveTrain\_Max;

//Serial.println(F("15 seconds forward"));

} else {//NumTouch == 1 or 2 or 3

if (NumTouch == 3) {

//Override the value of durationMoveTrain

durationMoveTrain = 1000; //durationMoveTrain = durationMoveTrain\_Max; //durationMoveTrain = 40000;

//Serial.println(F("1 second backward"));

} else {//NumTouch == 1 or 2

if (NumTouch == 1) {

//Override the value of durationMoveTrain

durationMoveTrain = NumTouch \* 1000; //durationMoveTrain = durationMoveTrain\_Max; //durationMoveTrain = 40000;

//Serial.println(F("1/2 seconds forward"));

} else {//NumTouch == 2

//Override the value of durationMoveTrain

durationMoveTrain = 5000; //durationMoveTrain = durationMoveTrain\_Max; //durationMoveTrain = 40000;

//Serial.println(F("1/2 seconds forward"));

}

}

}

//Serial.println(durationMoveTrain);

//to put the variable below before the while loop is the same//byte isFirst = 1;

step = 1;

}

}

}

}

}

} else {// (NumTouch >= 7 or == 0

if (NumTouch >= 7) {

//Serial.println(F("button7/8/9"));

//switch

//switch between the original reference point and new reference point

if (NumTouch == 7) {

//set the joint point as the reference point

referencePoint = jointPoint;

//durationMoveTrain\_referPoint2jointPoint was the basis, it is 0 now

durationMoveTrain\_referPoint2jointPoint = 0;

initialization();

isRunTime = true;

NumTouch = 0;

} else {

if (NumTouch == 8) {

//set 145 as the reference point

referencePoint = 145;

durationMoveTrain\_referPoint2jointPoint = durationMoveTrain\_OriginalReferPoint2jointPoint;

initialization();

isRunTime = true;

NumTouch = 0;

} else {//NumTouch >= 9

//skip

}

}

} else {//NumTouch == 0

//default

//Serial.println(F("button0"));

initTrain(sensor1);

step = 0;

}

}

}

}

}

//========================================================================================================================================================================

void loopInSetup() {

//simplified loop() {}

//corner case: cannot detect the barrier

//if there is barrier between sensor1 and train, train will moveforward without stop

isRunTime = true;

//for (unsigned short i = 65535; i > 0; i--) {}//this is not 65536, which causes overflow.

while(1) {

//initial

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//initialization();

currentMillis = millis();

// capture the latest value of millis()

// this is equivalent to noting the time from a clock

// use the same time for all operators to keep them synchronized

// in order to synchronize operators in each iteration of loop() {}, such as robot arm (servro1-6), a train (servo7)

//read

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

readRobot();

//Receive data returned by the robotic arm

//asynchronous, fast

//delay(100); // the TOF distance sensors need 90 ms to read

read\_multi\_tof\_sensors();

//analysis

//make decisions

//take actions

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//initalize operators

//Step 0.default position:

//servos 1 - 6 stand still/ expended arm position (group action 0), the arm near the fridge by servos 7 rotation (initTrain())

robotController.runActionGroup(0, 1);

//initialize train

initTrain(sensor1);

if (!isRunningTrain) {break;}

delay(trainDuration);

}

isRunTime = false;

}

//Step 1.

//mian function:throw the leftover away:

//========================================================================================================================================================================

bool step1() {

static bool isDone = false;

unsigned short robotInterval = trainDuration;

static unsigned long previousButtonMillis\_timer; //unsigned long

static byte step;

//Serial.println(F("step1().step"));

//Serial.println(step);

if (previousButtonMillis\_timer <= abs2(currentMillis - (long)robotInterval)) {

previousButtonMillis\_timer += robotInterval;

if (step == 0) {

//train move forward (to the right side on the picture), move to the front of the table,

//durationMoveTrain = durationMoveTrain\_table;

//NumTouch = 6;

if(NumTouch != 10)

//the check process above is for the setup() or initialization process

return false;

if (isHereArray != 0x00 && isSafe2grib) {

isSafe2grib = 0;

//Serial.println(F("in1"));

//Serial.println(durationMoveTrain);

gribBowl();

} else {//isHereArray == 0x00 || !isSafe2grib

if (isHereArray == 0x00 && !isSafe2grib) {

//even if is not running, it could be unsafe //isSafe2grib = 1;

if (!robotController.isRunning()) {

//Serial.println(F("in2"));

//Serial.println(durationMoveTrain);

step = 1;

//move it to the trash bin, moveForward(durationMoveTrain\_trashBin++)

durationMoveTrain = abs3(durationMoveTrain\_trashBin - durationMoveTrain\_table); //abs3() only applied to the integer

//isSafe2grib = 0;

}

//Serial.println(F("in3"));

//Serial.println(durationMoveTrain);

}

//Serial.println(F("in4"));

//Serial.println(durationMoveTrain);

}

//Serial.println(F("in5"));

// Serial.println(durationMoveTrain);

} else {

if (step == 1) {

moveForwardSafe();

if (!isRunningTrain) {

isSafe2grib = 1;

step = 2;

}

} else {

if (step == 2) {

if (isSafe2grib) {

isSafe2grib = 0;

throwLeftover();

} else {//!isSafe2grib

if (!robotController.isRunning()) {

step = 3;

//move back(to the left side) to the table,

durationMoveTrain = abs3(durationMoveTrain\_trashBin - durationMoveTrain\_table) + 2000; //recalculate the variance of change direction

//abs3() only applied to the integer

}

}

} else {

if (step == 3) {

moveBackwardSafe();

if (!isRunningTrain) {

isSafe2grib = 1;

step = 4;

}

} else {

if (step == 4) {

if (isSafe2grib) {

isSafe2grib = 0;

//put bowl on the table, group action 57

robotController.runActionGroup(57, 1);

} else {//!isSafe2grib

if (!robotController.isRunning()) {

step = 5;

}

}

} else {

if (step == 5) {

//go back to the default position, initTrain(),(group action 0)

//initTrain(sensor1); //NumTouch = 0;

//robotController.runActionGroup(0, 1);

//only if we make sure there is no barrier, we can use NumTouch = 7; //NumTouch = 8;

//count++;

//if (count >= )

previousButtonMillis\_timer += 1 \* 24 \* 60 \* 60 \* 1000; //set clock

//step = 0;

step = 6;

//default go back to step 0

isSafe2grib = 1;

isDone = true;

return isDone;

} else {//step >= 6 //others

//skip

//previousButtonMillis\_timer += 1 \* 24 \* 60 \* 60 \* 1000; //set clock

//step = 0;

//default go back to step 0

}

}

}

}

}

}

return isDone;

}

}

//Step 3.feed:

//========================================================================================================================================================================

void step3() {

//do not use switch(){} in the if() {}{} structure. No matter directly or indirectly call in the if() {}, switch() will be not working

unsigned short robotInterval = trainDuration;

static unsigned long previousButtonMillis\_timer; //unsigned long

static byte step;

//Serial.println(F("step3().step"));

//Serial.println(step);

//only each action in each step, because multi action cause some strange conflicts( such as only run one of them, or miss to check some of them)

//when moving train, no use arm group action, because in each step, the action is checked several times

if (previousButtonMillis\_timer <= abs2(currentMillis - (long)robotInterval)) {

previousButtonMillis\_timer += robotInterval;

if (step == 0) {

if(NumTouch != 10)//assume the robot is in front of table

//the check process above is for the setup() or initialization process

return;

if (isHereArray == 0x00)

//!!!!! after test, changeto isHereArray != 0x00

return;

if (!robotController.isRunning()) {

step = 22;

//train move backward 2s //move to the side of the fidge(joint point + 9s , moveForward(durationMoveTrain\_fridgeSide),

durationMoveTrain = 2000;

}

} else {

if (step == 22) {

moveBackwardSafe();

if (!isRunningTrain) {

isSafe2grib = 1;

step = 19;

}

} else {

if (step == 19) {

if (isSafe2grib) {

isSafe2grib = 0;

// (group 51)open the door of the fridge, group action 51, //1s 621 + 1s (642) + 1s (653) + 1s (674)+ fast rotota 1s(skip695, rotate servo 6 to 716 or 737)

//run group action 51 twice

robotController.runActionGroup(51, 1);

//assume the runActionGroup runtime of the robot arm is much longer than that of runTrain

} else {//isHereArray != 0x00 && !isSafe2grib //isSafe2grib = 0;

if (!robotController.isRunning()) {

//even if is not running, it could be unsafe

step = 20;

isSafe2grib = 1; //isSafe2grib = 1;

}

}

} else {

if (step == 20) {

if (isSafe2grib) {

isSafe2grib = 0;

//open the door of the fridge, group action 51, //1s 621 + 1s (642) + 1s (653) + 1s (674)+ fast rotota 1s(skip695, rotate servo 6 to 716 or 737)

//update 59 is 51\_2 new, sweep a little more then first time

//which arm can avoid to be hindered by the line when run group 55

robotController.runActionGroup(59, 1);

//if write this sentence in one step, actual run only once,

//besides, there is a problem about isRunning return

} else {//!isSafe2grib

if (!robotController.isRunning()) {

step = 1;

//train move backward 1s

durationMoveTrain = 1000;

}

}

} else {

if (step == 1) {

moveBackwardSafe();

if (!isRunningTrain) {

isSafe2grib = 1;

step = 2;

}

} else {

if (step == 2) {

if (isSafe2grib) {

isSafe2grib = 0;

//group action 55: push the door open

robotController.runActionGroup(55, 1);

} else {//!isSafe2grib

if (!robotController.isRunning()) {

step = 3;

//train move backward 8s

durationMoveTrain = 8 \* 1000;

}

}

} else {

if (step == 3) {

moveBackwardSafe();

if (!isRunningTrain) {

isSafe2grib = 1;

step = 4;

}

} else {

if (step == 4) {

if (isSafe2grib) {

isSafe2grib = 0;

//group action 56: push the door open, back and forth

robotController.runActionGroup(56, 1);

} else {//!isSafe2grib

if (!robotController.isRunning()) {

step = 5;

isSafe2grib = 1;

}

}

} else {

if (step == 5) {

if (isSafe2grib) {

isSafe2grib = 0;

//pull arm back group action 58 never use group action 0:initial arm here

robotController.runActionGroup(58, 1);

} else {//!isSafe2grib

if (!robotController.isRunning()) {

step = 6;

//train move backward,moveBackward(durationMoveTrain\_fridge\_sensor) or initTrain()

}

}

} else {

if (step == 6) {

//train move backward,moveBackward(durationMoveTrain\_fridge\_sensor) or initTrain()

initTrain(sensor1);

if (!isRunningTrain) {

isSafe2grib = 1;

step = 7;

}

} else {

if (step == 7) {

if (isSafe2grib) {

isSafe2grib = 0;

//and then grib the container pillet, (need to make sure where is the bowl (how far = distance between each isHere sensor to the robot arm,

//height and width is almost fixed), GROUP action 18 19 20, keep it close to the base before moving

robotController.runActionGroup(18, 1);

} else {//!isSafe2grib

if (!robotController.isRunning()) {

step = 8;

//move it to the side of table,moveForward(durationMoveTrain\_table), + 12s

durationMoveTrain = durationMoveTrain\_table + 12 \* 1000;

}

} } else {

if (step == 8) {

moveForwardSafe();

if (!isRunningTrain) {

isSafe2grib = 1;

step = 9;

}

} else {

if (step == 9) {

if (isSafe2grib) {

isSafe2grib = 0;

//rotate the container several times quickly over the bowl,(how far = distance between each isHere sensor to the robot arm,

//height and width is almost fixed, 61 62 63 64 65 66 67), group action\_turnOver() 7,keep pillet close to the base and

robotController.runActionGroup(61, 1);

} else {//!isSafe2grib

if (!robotController.isRunning()) {

step = 10;

//moveBackward(durationMoveTrain\_fridge\_sensor) or initTrain() 8 9 10

}

}

} else {

if (step == 10) {

//moveBackward(durationMoveTrain\_fridge\_sensor) or initTrain() 8 9 10

initTrain(sensor1);

if (!isRunningTrain) {

isSafe2grib = 1;

step = 11;

}

} else {

if (step == 11) {

if (isSafe2grib) {

isSafe2grib = 0;

//put the pillet and cracker container back, group action 6,

robotController.runActionGroup(6, 1);

} else {//!isSafe2grib

if (!robotController.isRunning()) {

step = 12;

isSafe2grib = 1;

}

}

} else {

if (step == 12) {

if (isSafe2grib) {

isSafe2grib = 0;

//close the door, group action 52,

robotController.runActionGroup(52, 1);

} else {//!isSafe2grib

if (!robotController.isRunning()) {

step = 21;

//besides, moveForward(durationMoveTrain\_fridge++)12s,

durationMoveTrain = 12 \* 1000;

}

}

} else {

if (step == 21) {

moveForwardSafe();

if (!isRunningTrain) {

isSafe2grib = 1;

step = 13;

}

} else {

if (step == 13) {

if (isSafe2grib) {

isSafe2grib = 0;

//then ,knock the door, group action 53

robotController.runActionGroup(53, 1);

} else {//!isSafe2grib

if (!robotController.isRunning()) {

step = 14;

isSafe2grib = 1;

//group action 0

}

}

} else {

if (step ==14) {

if (isSafe2grib) {

isSafe2grib = 0;

robotController.runActionGroup(0, 1);

} else {//!isSafe2grib

if (!robotController.isRunning()) {

step = 15;

//go back to the default position, initTrain()

//go back to the default position, initTrain()

}

}

} else {

if (step == 15) {

//go back to the default position, initTrain()

//NumTouch = 0;

if (initTrain(sensor1)){ //if (!isRunningTrain) {

//only if we make sure there is no barrier, we can use NumTouch = 7; //NumTouch = 8;

//count++;

//if (count >= )

previousButtonMillis\_timer += 1 \* 24 \* 60 \* 60 \* 1000; //set clock

step = 16;

//default go back to step 0

}

} else {//step (>= 16 && <= 18) or (>= 23) //others

//skip

//previousButtonMillis\_timer += 1 \* 24 \* 60 \* 60 \* 1000; //set clock

//step = 0;

//default go back to step 0

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

//========================================================================================================================================================================

void readRobot() {

unsigned short robotInterval = 100;

static unsigned long previousButtonMillis\_timer; //unsigned long

if (previousButtonMillis\_timer <= abs2(millis() - (long)robotInterval)) {// currentMillis = abs2(millis() - (long)robotInterval)

previousButtonMillis\_timer += robotInterval;

robotController.receiveHandler();

}

}

//put it into the bin after the cap open, (need to make sure where is the bin (how

//far = distance between bin to the robot arm, it could be fixed as well, height and width

//is almost fixed), shake in front of the bin and pull arm back, turnOver = servo 1 rotate 90 degree

//hold the bowl, (group action 54)

//========================================================================================================================================================================

void throwLeftover() {

//unsigned short robotInterval = trainDuration;

//static unsigned long previousButtonMillis\_timer; //unsigned long

//if (previousButtonMillis\_timer <= abs2(currentMillis - (long)robotInterval)) {

//previousButtonMillis\_timer += robotInterval;

robotController.runActionGroup(54, 1);

//}

}

//gribbing the used bowl,(need to make sure where is the bowl (how far = distance

//between each isHere sensor to the robot arm, height and width is almost fixed),

//GROUP action 11 12 13 14 15 16 17, holding the bowl, keep it close to the base of arm before moving

//========================================================================================================================================================================

void gribBowl() {

unsigned short robotInterval = trainDuration;

static unsigned long previousButtonMillis\_timer; //unsigned long

if (previousButtonMillis\_timer <= abs2(currentMillis - (long)robotInterval)) {

previousButtonMillis\_timer += robotInterval;

//isHereArray recording the position of object

//Serial.println(F("isHereArray"));

//Serial.println(isHereArray);

if(isHereArray == 0x00) {

//case 0x00: //default//0x00(0000 0000)

//no object detected in front of sensors

//skip

} else {

if(isHereArray == 0x01) {

//case 0x01://0x01(0000 0001)

//just in front of the farthest InfraRedSensor

robotController.runActionGroup(11,1);

}else {

if(isHereArray == 0x03) {

//case 0x03://0x03(0000 0011)

//between the middle and the farthest InfraRedSensor

robotController.runActionGroup(12,1);

} else {

if(isHereArray == 0x02) {

//case 0x02://0x02(0000 0010)

robotController.runActionGroup(13,1);

} else {

if(isHereArray == 0x07) {

//case 0x07://0x07(0000 0111)

robotController.runActionGroup(13,1);

//just in front of the middle InfraRedSensor

} else {

if(isHereArray == 0x06) {

//case 0x06://0x06(0000 0110)

//between the closest and the middle InfraRedSensor

robotController.runActionGroup(14,1);

} else {

if(isHereArray == 0x04) {

//case 0x04://0x02(0000 0100)

robotController.runActionGroup(15,1);

} else {

if(isHereArray == 0x0E) {

//case 0x0E://0x02(0000 1110)//14

robotController.runActionGroup(15,1);

//just in front of the closest InfraRedSensor

} else {

if(isHereArray == 0x0C) {

//case 0x0C://0x0C(0000 1100)//12

//between the closest InfraRedSensor and the ultrasound sensor

robotController.runActionGroup(16,1);

} else {

if(isHereArray == 0x08) {

//case 0x08://0x08(0000 1000)

robotController.runActionGroup(17,1);

//just in front of the ultrasound sensor

} else {

//default://otherwise

//others shake the arm let pet leave the place

}

}

}

}

}

}

}

}

}

}

}

}

//========================================================================================================================================================================

void try2Recover() {

//sensor1 < sensor1\_Min || durationMoveTrain > durationMoveTrain\_Max

//normally it is not allowed to move over durationMoveTrain\_Max or under sensor1\_Min

//no sensor signal available

//but let us try to go back to the reference point, sensor1 145

//move train only if the sensor value is from sensor1 0 to 710

//protection:

//moveForward(durationMoveTrain\_Max); allowed max distance //moveForward(durationMoveTrain\_referPoint2jointPoint); in the setup() which means sensor1 310 joint point

//previousDistanceSensor1 = sensor1; //update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

//sensor1 sensor1\_Min allowed min distance

//reference point: 145 (143 - 147 is fine)

//trainDuration = (147 - 143)/ speedTrain = 595.24 ms

short temp = sensor1 - referencePoint; //unsigned short

//Serial.println(F("sensor1 - referencePoint"));

//Serial.println(temp);

//durationMoveTrain = (targetedPoint - referencePoint) / speedTrain;

durationMoveTrain = (int)(abs2((long)temp)/speedTrain); //abs3() can only be used for integer //unsigned int

//Serial.println(F("durationMoveTrain"));

//Serial.println(durationMoveTrain);

if (durationMoveTrain > durationMoveTrain\_Max && sensor1 <= 710) {

durationMoveTrain = durationMoveTrain\_referPoint2jointPoint;

allow2updatePreviousMoveServo7\_timer = true; //unlock

moveBackward(durationMoveTrain);

//not update// it is not correct //previousDistanceSensor1 = sensor1; //update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

//difference

} else {

if (sensor1 >= 0 && sensor1 < sensor1\_Min) {

durationMoveTrain = (referencePoint - sensor1\_Min)/ speedTrain; ////assume referencePoint > sensor1\_Min

allow2updatePreviousMoveServo7\_timer = true; //unlock

moveForward(durationMoveTrain);

} else {

// (sensor1 < 0 || sensor > 710) // too close || over far

//genearlly it is impossible to reach that close or far, meaning it could be some bugs or noise

//skip or ignore

}

}

}

//========================================================================================================================================================================

/\*

if (previousButtonMillis\_timer <= abs2(millis() - (long)buttonInterval)) {// currentMillis = abs2(millis() - (long)buttonInterval)

//if (digitalRead(buttonPin) != LOW) {

//delay(80);

if (digitalRead(buttonPin) == LOW) {

previousButtonMillis\_timer += buttonInterval;

Serial.println(F("button"));

Serial.println(NumTouchLocal);

NumTouchLocal++;

if (NumTouchLocal > 2) {NumTouchLocal = 1;}

//buttonLed\_State = ! buttonLed\_State; // this changes it to low if it was high

// and to high if it was low

}

//}

}

\*/

void readButton() {

//do not use while() loop or delay() in the loop() {}

//use synchronized timer/counter or asynchronous timer/counter

//set same clock or different clocks

// ask

// this only reads the button state after the button interval has elapsed

// this avoids multiple flashes if the button is pressed

// Notice that there is no need to synchronize this, using millis() instead of currentMillis

// use of millis() with the flashing Leds

// every time the button is pressed it changes buttonLed\_State causing:

static byte step; //0 - 255

static byte NumTouchLocal = 0;

static unsigned short count; //0 - 65535

static unsigned long previousButtonMillis\_timer;

unsigned short buttonInterval = 200; //200; //300

//if (robotController.isRunning())

// return;

//if (isRunningTrain)

// return;

if (previousButtonMillis\_timer <= abs2(millis() - (long)buttonInterval)) {// currentMillis = abs2(millis() - (long)buttonInterval)

if (step == 0) //do not use switch(){} in the if() {}{} structure

{//case 0:

if (digitalRead(buttonPin) == LOW)

{//Detection touch sensor

digitalWrite(LED, HIGH);

delay(500);

if (digitalRead(buttonPin) != LOW)

{//If it is short press once

digitalWrite(LED, LOW);

NumTouchLocal ++ ;

count = 0;

step = 1;

previousButtonMillis\_timer += buttonInterval;

}

}

} else {

if (step == 1) {

//case 1:

if (digitalRead(buttonPin) == LOW)

{//Detection touch sensor

digitalWrite(LED, HIGH);

delay(500);

if (digitalRead(buttonPin) != LOW)

{//If it is short touch once

NumTouchLocal ++ ;

digitalWrite(LED, LOW);

if (NumTouchLocal > 9)

NumTouchLocal = 9;

count = 0;

previousButtonMillis\_timer += buttonInterval;

}

}

else

{

count++;

if (count > 15)

{

step = 2;

count = 0;

}

else

{

previousButtonMillis\_timer += buttonInterval;

}

}

} else {

if (step == 2) {

//case 2:

//output NumTouchLocal

NumTouch = NumTouchLocal;

count++;

if (count > 1)

{

count = 0;

NumTouchLocal = 0;

step = 0;

}

} else {

//default:

NumTouchLocal = 0;

count = 0;

step = 0;

}

}

}

Serial.println(F("button"));

Serial.println(NumTouchLocal);

//Serial.println(NumTouch);

}

}

//========================================================================================================================================================================

bool isHere()

{ //ultrasound sensor

static unsigned long previousMillis\_timer;

unsigned short sensorInterval = 100;

bool isHere = false;

if (previousMillis\_timer <= abs2(millis() - (long)sensorInterval))// currentMillis - sensorInterval could be != abs2(millis() - (long)sensorInterval)

{

previousMillis\_timer += sensorInterval; //synchronize, besides, set clock

distance=((short)ultrasound.GetDistance())/10; //unsigned short

//Serial.println(distance); //Get and print distance of serial port, unit mm

//if (robotController.isRunning() == false) //Execute when the robotic arm stop running

{

if (distance <= 10)

{

isHere = true;

ultrasound.rainbow\_color();

} else {//>10

ultrasound.Color(0, 255, 0, 0, 255, 0); //green

}

}

return isHere;

}

}

//========================================================================================================================================================================

bool isHere(byte ID) //InfraRedSensor

{ //infrared sensor

//??digitalRead() is not compitable with timer/counter

//static unsigned long previousMillis\_timer;

//unsigned short sensorInterval = 500;

bool isHere = false;

//if (previousMillis\_timer <= abs2(millis() - (long)sensorInterval))// currentMillis - sensorInterval could be != abs2(millis() - (long)sensorInterval)

//{

//previousMillis\_timer += sensorInterval; //synchronize, besides, set clock

//a object is detected by the sensor

if (digitalRead(ID) == LOW)

{

isHere = true;

} else {

//skip

}

return isHere;

//}

}

//========================================================================================================================================================================

void setID() {

// all reset

digitalWrite(SHT\_LOX1, LOW);

digitalWrite(SHT\_LOX2, LOW);

digitalWrite(SHT\_LOX3, LOW);

digitalWrite(SHT\_LOX4, LOW);

delay(10);

// all unreset

digitalWrite(SHT\_LOX1, HIGH);

digitalWrite(SHT\_LOX2, HIGH);

digitalWrite(SHT\_LOX3, HIGH);

digitalWrite(SHT\_LOX4, HIGH);

delay(10);

// activating LOX1 and reseting LOX2

digitalWrite(SHT\_LOX1, HIGH);

digitalWrite(SHT\_LOX2, LOW);

digitalWrite(SHT\_LOX3, LOW);

digitalWrite(SHT\_LOX4, LOW);

/\*

// initing LOX1

Wire.begin(); //This function initializes the Wire library

lox1.setAddress(LOX1\_ADDRESS); //set address//

if(!lox1.init()) {

Serial.println(F("Failed to boot first VL53L0X"));

//while(1);

}

\*/

// initing LOX1

if(!lox1.begin(LOX1\_ADDRESS, false, &Wire, 3)) {

//Serial.println(F("Failed to boot first VL53L0X, LOX1"));

//while(1);

}

delay(10);

// activating LOX2

digitalWrite(SHT\_LOX2, HIGH);

delay(10);

//initing LOX2

//Wire.begin(); //This function initializes the Wire library

//default//lox2.setAddress(LOX2\_ADDRESS); //set address//

//if(!lox2.init()) {

// Serial.println(F("Failed to boot second VL53L0X"));

//while(1);

//}

// activating LOX3

digitalWrite(SHT\_LOX3, HIGH);

delay(10);

//initing LOX3

Wire.begin(); //This function initializes the Wire library

lox3.setAddress(LOX3\_ADDRESS); //set address//

if(!lox3.init()) {

//Serial.println(F("Failed to boot third VL53L0X, LOX3"));

//while(1);

}

//lox3.setMeasurementTimingBudget(sensor1\_Min);

//a longer timing budget allows for more accurate measurements.

// activating LOX4

digitalWrite(SHT\_LOX4, HIGH);

delay(10);

//initing LOX4

Wire.begin(); //This function initializes the Wire library

lox4.setAddress(LOX4\_ADDRESS); //set address//

if(!lox4.init()) {

//Serial.println(F("Failed to boot fourth VL53L0X, LOX4"));

//while(1);

}

}

//========================================================================================================================================================================

void read\_multi\_tof\_sensors() {

// the TOF distance sensors need 90 ms to read

//asynchronous, fast

static unsigned long previousMillis\_timer;

unsigned short sensorInterval = 100; //short

if (previousMillis\_timer < abs2(millis() - (long)sensorInterval))// currentMillis - sensorInterval could be != abs2(millis() - (long)sensorInterval)

{

previousMillis\_timer += sensorInterval; //set clock

//make sure that the sensor value is number instaed of some mysterious string

// this holds the measurement

VL53L0X\_RangingMeasurementData\_t measure1;

lox1.rangingTest(&measure1, false); // pass in 'true' to get debug data printout!

// print sensor1 reading

Serial.print(F("1: "));

if(measure1.RangeStatus != 4) { // if not out of range

sensor1 = measure1.RangeMilliMeter;

Serial.print(sensor1);

//Serial.print(F("mm"));

} else {

//Serial.print(F("Out of range"));

}

/\*

Serial.print(F("1: "));

Serial.print(lox1.readRangeSingleMillimeters());

Serial.print(F("mm"));

\*/

//Serial.print(F(" "));

// print sensor2 reading

//Serial.print(F("2: "));

// Serial.print(lox2.readRangeSingleMillimeters());

// Serial.print(F("mm"));

Serial.print(F(" "));

// print sensor3 reading

Serial.print(F("3: "));

sensor3 = lox3.readRangeSingleMillimeters();

Serial.print(sensor3);

//Serial.print(F("mm"));

Serial.print(F(" "));

// print sensor4 reading

Serial.print(F("4: "));

sensor4 = lox4.readRangeSingleMillimeters();

Serial.print(sensor4);

//Serial.print(F("mm"));

Serial.println();

}

}

//========================================================================================================================================================================

bool isHinderedSensor1() {

//asynchronous

//static unsigned long previousMillis\_timer;

//unsigned short sensorInterval = 100;

bool isHindered = false; //sensor laser beam is not hindered

//unsigned long difference;

// if (previousMillis\_timer <= abs2(millis() - (long)sensorInterval))// currentMillis - sensorInterval could be != abs2(millis() - (long)sensorInterval)

//{

// previousMillis\_timer += sensorInterval; //set clock // previousMillis\_timer = previousMillis\_timer + sensorInterval;

//double check whether we need to update the durationMoveTrain

//sensor1 is not stuck

//assume the sensor1 value will not stuck in a constant value when the robot arm moves

//assumw the pet will not stand between the sensor and the robot arm

//delay(100);

//sensor1 has been updated

//previousDistanceSensor1 is still the same as above in the function

//952 > 100 no need to update previousMillis\_timer//1000; //no need 120;

if (isRunningTrain) { //train is moving

short temp = previousDistanceSensor1 - sensor1;

//Serial.println(F("previousDistanceSensor1"));

//Serial.println(previousDistanceSensor1);

//Serial.println(F("sensor1"));

//Serial.println(sensor1);

//Serial.println(F("difference"));

//Serial.println((short)abs2((long)temp)); //abs3() can only be used for integer

//Serial.println(F("2 \* trainDuration \* speedTrain"));

//Serial.println(2 \* trainDuration \* speedTrain);

//if is hindered, the difference > 0

if ((previousDistanceSensor1 > sensor1) && (short)abs2((long)temp) > barrierIndex \* trainDuration \* speedTrain) { // (referencePoint + 2 - sensor1\_Min) = 47

//abs3() can only be used for integer

//difference > 2 \* trainDuration \* speedTrain

//case1: sensor1 is hindered by pet or something else

//because of sensor stuck

//due to the barrier

isHindered = true;

} else {//difference

//skip

}

} else {//!isRunningTrain

//corner case

//skip

//

}

//Serial.println(F("isHindered"));

//Serial.println(isHindered);

return isHindered;

//}

}

//========================================================================================================================================================================

//Moving train to the reference point is its function

//moveBackward(), if the robot is on the right side of the reference point

//moveForward(), if the robot is on the left side of the reference point

bool initTrain(unsigned short sensor1) {

//moving the robot arm platform to the reference point is the function

//normally, it is used in the runtime, loop() {}

//initial

static unsigned long previousMillis\_timer; //set loop check clock

static byte isFirstTime = 1; //flag check whether it is the first time to enter the codes about the hindered state

bool isDone = false;

if(previousMillis\_timer <= abs2(currentMillis - (long)trainDuration))

{

previousMillis\_timer += trainDuration;

//synchronize, besides, set clock

//1000; //no need 120;

//Serial.println(F("sensor1"));

//Serial.println(sensor1);

//Serial.println(F("previousDistanceSensor1"));

//Serial.println(previousDistanceSensor1);

if (sensor1 >= referencePoint - 2 && sensor1 <= referencePoint + 2) {//about 145

//stop

isDone = true;

stopTrain();

//it could be wrong// previousDistanceSensor1 = sensor1;

return isDone;

} else {// sensor1 < referencePoint - 2 || sensor1 > referencePoint + 2

//Serial.println(F("sensor1 < referencePoint - 2 || sensor1 > referencePoint + 2"));

//move train only if the sensor value is from sensor1 0 to 710

//protection:

//moveForward(durationMoveTrain\_Max); allowed max distance //moveForward(durationMoveTrain\_referPoint2jointPoint); in the setup() which means sensor1 310 joint point

//previousDistanceSensor1 = sensor1; //update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

//sensor1 sensor1\_Min allowed min distance

//reference point: 145 (143 - 147 is fine)

//trainDuration = (147 - 143)/ speedTrain = 595.24 ms

short temp = sensor1 - referencePoint; //unsigned short

//Serial.println(F("sensor1 - referencePoint"));

//Serial.println(temp);

//durationMoveTrain = (targetedPoint - referencePoint) / speedTrain;

durationMoveTrain = (int)(abs2((long)temp)/speedTrain); //abs3() can only be used for integer //unsigned int

//Serial.println(F("durationMoveTrain"));

//Serial.println(durationMoveTrain);

if (sensor1 >= sensor1\_Min && durationMoveTrain <= durationMoveTrain\_Max) {

//Serial.println(F("isHindered: "));

//Serial.println(isHinderedSensor1());

if (isHinderedSensor1()) {

//assume the train is moving

//case1: sensor1 is hindered by pet or something else

//the reference point calculated during the runtime is wrong due to barrier

//but it is fine, after the pet move away, the reference pint will be recalculated and corrected.

//Serial.println(F("isB"));

if (isFirstTime) {//step == 1

//normally, the durationMoveTrain(remainning duration) is recorded only (once) at the beginning of movement

//during the movement, the previousMoveServo7\_timer is not updated

//if eveything is running smoothly, the durationMoveTrain is equal to the difference between the previousMoveServo7\_timer recorded and currentTime

//if the train is paused temperarily, besides, the real time(currentTime) is never stopped

// we recorded the remainning duration only (once) at the beginning of the pause (not before the robot moves again, because the currentTime is changed).

//therefore, we can assign the remainning duration to durationMoveTrain only (once) at the beginning of the pause, like this:

///////////durationMoveTrain = abs3(durationMoveTrain\_remainingPlanned - trainDuration);

//durationMoveTrain\_accumulationActual = abs2(currentMillis - previousMoveServo7\_timer);

// durationMoveTrain's physical rule is recording the total/remaining time of train movement.

//which we can use the same variable, durationMoveTrain, to represent two physical rules:

//1. Represent the total time point when there is no barrier. In other words, planned duration of the train Movement.

//////////durationMoveTrain = targeted duration; //For example, durationMoveTrain = 5000; //meaning set 5s movement

//////////allow2updatePreviousMoveServo7\_timer = true; //unlock

//2. Represent the remainning time period when there is a barrier.

//////////durationMoveTrain = durationMoveTrain\_remainingPlanned;

//////////allow2updatePreviousMoveServo7\_timer = true; //unlock

durationMoveTrain = abs3(durationMoveTrain\_remainingPlanned - abs2(millis() - previousMillis)); //?0.5 delay

//durationMoveTrain\_accumulationActual = abs2(currentMillis - previousMoveServo7\_timer);

//the durationMoveTrain\_remainingPlanned will be recorded by durationMoveTrain.

//abs3() only applied to the integer

//abs3() can only be used for integer

allow2updatePreviousMoveServo7\_timer = true; //unlock

isFirstTime2SetDurationMoveTrain\_remainingPlanned = true;

isFirstTime = 0; // step = 2;

//pause

pauseTrain();

//it could be wrong// previousDistanceSensor1 = sensor1;

//assume the train is moving, just have to wait for pet leaving there.

/\*

if (isRunningTrain) {

} else {//!isRunningTrain

}

\*/

//there is barrier, the sensor1 value is wrong

//previousDistanceSensor1 = sensor1;

//update it only if distance is correct which means no barrier

} else {//!isFirstTime

//skip

//wait here

//pause

pauseTrain();

//it could be wrong// previousDistanceSensor1 = sensor1;

//assume the train is moving, just have to wait for pet leaving there.

//there is barrier, the sensor1 value is wrong

//previousDistanceSensor1 = sensor1;

//update it only if distance is correct which means no barrier

}

} else {//!isHinderedSensor1()

isFirstTime = 1; // step = 1;

if (isFirstTime2SetDurationMoveTrain\_remainingPlanned) {

isFirstTime2SetDurationMoveTrain\_remainingPlanned = false;

//1.only (once) at the beginning of train movement (before recover from the hindered state),

//2.only (once) at the beginning of train movement (user set),

//the durationMoveTrain\_remainingPlanned will be reset

//only the train is really stopped for a reasonable reason, not pause not skip, just correct to stop

//In other words, the train finished the request movement, then

//isFirstTime2SetDurationMoveTrain\_remainingPlanned = true;

durationMoveTrain\_remainingPlanned = durationMoveTrain;

} else {

durationMoveTrain\_remainingPlanned = abs3(durationMoveTrain\_remainingPlanned - abs2(millis() - previousMillis));

//the value above which is calculated since the very second time. In other words,

//this variable will be updated (all the time) during the required movement

//abs3() can only be used for integer

}

previousMillis = millis();

//durationMoveTrain can be used to show the big picture, while it could not be accurate when train is far from joint point

if (sensor1 > jointPoint && durationMoveTrain <= durationMoveTrain\_Max) {

//synchronize

//Serial.println(F("currentMillis"));

//Serial.println(currentMillis);

//Serial.println(F("previousMoveServo7\_timer"));

//Serial.println(previousMoveServo7\_timer);

//Serial.println(F("durationMoveTrain\_remainingPlanned"));

//Serial.println(durationMoveTrain\_remainingPlanned);

//Serial.println(F("durationMoveTrainBeforeOverride"));

//Serial.println(durationMoveTrain);

//check the difference between scheduled (the previous loop) remaining and actual (the currunt loop)

//if the variance between differenceDuration and trainDuration is too large, meaning the speed is very unstable or the sensor1 is very unaccurate

unsigned int differenceDuration = abs3(durationMoveTrain\_remainingPlanned - durationMoveTrain); //abs3() can only be used for integer

//Serial.println(F("differenceDuration"));

//Serial.println(differenceDuration);

//Serial.println("trainDuration");

//Serial.println(trainDuration);

//Through the test, we found it is because the sensor1 is not accurate in this case

//Therefore, give up to control precisely the position through the sensor1 in the case

//Override the value of durationMoveTrain

durationMoveTrain = durationMoveTrain\_OriginalReferPoint2jointPoint;

//Serial.println(durationMoveTrain);

// corner case if the runtime is smaller than the calculated durationMoveTrain at the very beginning, settng clock will be missed

//check the difference between scheduled (the previous loop) remaining and actual (the currunt loop)

if (durationMoveTrain\_remainingPlanned < durationMoveTrain) {

// (abs3(durationMoveTrain\_remainingPlanned) < durationMoveTrain)//set clock ////abs3() can only be used for integer

//abs3() can only be used for integer

//Serial.println(F("bigger asdfsdsff"));

//skip or continue to move

//small deviation or vibration is acceptable

//update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

if (previousDistanceSensor1 > sensor1) {

previousDistanceSensor1 = sensor1;

} else {//previousDistanceSensor1 <= sensor1

//skip

}

} else {//durationMoveTrain\_remainingPlanned >= durationMoveTrain

//Serial.println(F("bigger jointPointfsdfsdfsdfsdf"));

allow2updatePreviousMoveServo7\_timer = true; //unlock

moveBackward(durationMoveTrain);

}

} else {

//sensor1\_Min <= sensor1 < referencePoint - 2 || referencePoint + 2 < sensor1 <= jointPoint

//case2: sensor1 is not accurate (the value is too big or too small) during the previous loop period

//accurate control

short temp = previousDistanceSensor1 - sensor1;

unsigned short differenceDistance = (short)abs2((long)temp);

//Serial.println(F("sensor1\_Min <= sensor1 < referencePoint - 2 || referencePoint + 2 < sensor1 <= jointPoint"));

//Serial.println(F("previousDistanceSensor1"));

//Serial.println(previousDistanceSensor1);

//Serial.println(F("differenceDistance"));

//Serial.println(differenceDistance);

//Serial.println(F("trainDuration \* speedTrain \* (1 - safeIndex)"));

//Serial.println(trainDuration \* speedTrain \* (1 - safeIndex));

//Serial.println(F("trainDuration \* speedTrain \* (1 + safeIndex)"));

//Serial.println(trainDuration \* speedTrain \* (1 + safeIndex));

//if (differenceDistance >= trainDuration \* speedTrain \* (1 - safeIndex) && differenceDistance <= trainDuration \* speedTrain \* (1 + safeIndex) ) {

//idealy, the speed of train is stable,

//the distance difference per duration = trainDuration \* speedTrain // (147 - 143) = 5 //10 //10.5

//Besides, considering the sensor value deviation.//but moveback value deviation is pretty small

//sensor1 value could be floating between expectedValue - 1 to expectedValue + 1

// (592 \* 0.0084 \* 0.7 = 3.4810, center = 592 \* 0.0084 = 5, 592 \* 0.0084 \* 1.3 = 6.4646)

// (592 \* 0.0084 \* 0.2 = 0.9946, center = 592 \* 0.0084 = 5, 592 \* 0.0084 \* 1.8 = 8.9510)

//nevertheless, the actual speed is a little high,

//592 \* 0.0105 = 6.2160, it is very close to the upper boundary,

//therefore, how about 0.7 as upper-band safeindex, 592 \* 0.0084 \* 1.7 = 8.4538

//test 3 use safeindex = 0.8 control both boundries

if (differenceDistance >= trainDuration \* speedTrain \* (1 - safeIndex) && differenceDistance <= trainDuration \* speedTrain \* (1.7) ) {

//Serial.println(F("less jointPoint durationMoveTrain\_Max0skip"));

//continue to move

//small deviation or vibration is acceptable

previousDistanceSensor1 = sensor1;

//update it only if distance is correct which means no barrier

} else { //it could stop (differenceDistance < trainDuration \* speedTrain \* (1 - safeIndex)) or not in the range

//reset

//Override the value of durationMoveTrain

if (durationMoveTrain < trainDuration) {

//1000; //define the smallest step distance//resolution

durationMoveTrain = trainDuration;

} else {

//skip

}

allow2updatePreviousMoveServo7\_timer = true;

if (sensor1 > referencePoint + 2) {

//Serial.println(F("sensor1 > referencePoint + 2 movef"));

//if (sensor1 > referencePoint + 2 && sensor1 <= jointPoint) {}//joint

moveBackward(durationMoveTrain); //set clock

} else {//< referencePoint - 2

//Serial.println(F("< referencePoint - 2moveForward"));

//if (sensor1 >= sensor1\_Min && sensor1 < referencePoint - 2) {}

moveForward(durationMoveTrain); //set clock ////case1:

}

}

}

}

} else {

//sensor1 < sensor1\_Min || durationMoveTrain > durationMoveTrain\_Max

//normally it is not allowed to move over durationMoveTrain\_Max or under sensor1\_Min

//no sensor signal available

//Serial.println(F("sensor1 < sensor1\_Min || durationMoveTrain > durationMoveTrain\_Max"));

pauseTrain();

if (sensor1 >= 0 && sensor1 < sensor1\_Min) {

//update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

if (previousDistanceSensor1 < sensor1) {

previousDistanceSensor1 = sensor1;

} else {//previousDistanceSensor1 >= sensor1

//skip

}

}

}

}

//Serial.println('loop');

return isDone; //-1//false//while(1);

}

}

//=======================================================================================================================================================================

unsigned int abs3(int temp) {

unsigned int result;

//traditional abs3() can only be used for integer

if (temp >= 0) {

result = temp;

} else {//temp <= 0

result = 0 - temp;

}

return result;

}

//=======================================================================================================================================================================

unsigned long abs2(long temp) {

unsigned long result;

//traditional abs3() can only be used for integer

if (temp >= 0) {

result = temp;

} else {//temp <= 0

result = 0 - temp;

}

return result;

}

//train move backward (to the left side):

//========================================================================================================================================================================

void moveBackward(unsigned int durationMoveTrain) {

if (isRunTime) {

static unsigned long previousMillis\_timer; //set loop check clock

if(previousMillis\_timer <= abs2(currentMillis - (long)trainDuration))

{

previousMillis\_timer += trainDuration;

//synchronize, besides, set clock

//1000; // no need 250;

//protection:

//moveForward(durationMoveTrain\_Max); allowed max distance //moveForward(durationMoveTrain\_referPoint2jointPoint); in the setup() which means sensor1 310 joint point

//previousDistanceSensor1 = sensor1; //update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

//sensor1 sensor1\_Min allowed min distance

//reference point: 145 (143 - 147 is fine)

if (sensor1 >= sensor1\_Min) {

//Serial.println(F("allow2updatePreviousMoveServo7\_timer"));

//Serial.println(allow2updatePreviousMoveServo7\_timer);

if (allow2updatePreviousMoveServo7\_timer) {

//start

//////records the previous time point only (once) at the beginning of train movement, like this:

//////////if (allow2updatePreviousMoveServo7\_timer == true) {}

//////keeps or locks the previous time point/value/variable during the whole train movement process:

//////////allow2updatePreviousMoveServo7\_timer = false;

//////////previousMoveServo7\_timer = currentMillis;

//reset the previousMoveServo7\_timer automatically only if the train will begin to move.

//There are two situations in which the train will begin to move:

// (a) when users ask to move train or

// (b) when the train is not hindered and it continues to finish the previous movement.

allow2updatePreviousMoveServo7\_timer = false; //lock

previousMoveServo7\_timer = currentMillis;

//Serial.println(F("previousMoveServo7\_timer"));

//Serial.println(previousMoveServo7\_timer);

//synchronize

// capture the latest value of currentMillis

// this is equivalent to noting the time from a clock

// use the same time for all servo 7 movements to keep them synchronized

robotController.moveServo(7, 485, 1000); //set speed

isRunningTrain = true;

} else {//!allow2updatePreviousMoveServo7\_timer

//Serial.println(durationMoveTrain\_accumulationActual);

//Serial.println(durationMoveTrain);

unsigned long durationMoveTrain\_accumulationActual = abs2(currentMillis - previousMoveServo7\_timer); //abs3() can only be used for integer

if (durationMoveTrain\_accumulationActual >= durationMoveTrain) {// only no sensor signal case, we guess

stopTrain();

//it could be wrong// previousDistanceSensor1 = sensor1;

} else {//durationMoveTrain\_accumulationActual < durationMoveTrain

//skip

//do nothing if we have already stopped the train

//continue to move if the train is running

}

}

//update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

if (previousDistanceSensor1 > sensor1) {

previousDistanceSensor1 = sensor1;

} else {//previousDistanceSensor1 <= sensor1

//skip

}

} else {//sensor < sensor1\_Min

//skip //do nothing

}

} else {

//skip//do nothing

}

} else {//set up

//protection:

//moveForward(durationMoveTrain\_Max); allowed max distance //moveForward(durationMoveTrain\_referPoint2jointPoint); in the setup() which means sensor1 310 joint point

//previousDistanceSensor1 = sensor1; //update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

//sensor1 sensor1\_Min allowed min distance

//reference point: 145 (143 - 147 is fine)

if (sensor1 >= sensor1\_Min) {

robotController.moveServo(7, 485, 1000); //set speed

/\*

//update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

if (previousDistanceSensor1 > sensor1) {

previousDistanceSensor1 = sensor1;

} else {//previousDistanceSensor1 <= sensor1

//skip

}

\*/

isRunningTrain = true;

delay(durationMoveTrain);

stopTrain();

//assume the sensor1 laser beam is not hindered at the very beginnings

//it could be wrong// previousDistanceSensor1 = sensor1;

} else {//sensor < sensor1\_Min

//skip //do nothing

}

}

}

//train move forward (to the right side on the picture):

//========================================================================================================================================================================

void moveForward(unsigned int durationMoveTrain) {

if (isRunTime) {

static unsigned long previousMillis\_timer; //set loop check clock

if(previousMillis\_timer <= abs2(currentMillis - (long)trainDuration))

{

previousMillis\_timer += trainDuration;

//synchronize, besides, set clock

//1000; // no need 250;

//protection:

//moveForward(durationMoveTrain\_Max); allowed max distance //moveForward(durationMoveTrain\_referPoint2jointPoint); in the setup() which means sensor1 310 joint point

//previousDistanceSensor1 = sensor1; //update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

//sensor1 sensor1\_Min allowed min distance

//reference point: 145 (143 - 147 is fine)

if (durationMoveTrain <= durationMoveTrain\_Max) {

//Serial.println(allow2updatePreviousMoveServo7\_timer);

if (allow2updatePreviousMoveServo7\_timer) {

//start

//////records the previous time point only (once) at the beginning of train movement, like this:

//////////if (allow2updatePreviousMoveServo7\_timer == true) {}

//////keeps or locks the previous time point/value/variable during the whole train movement process:

//////////allow2updatePreviousMoveServo7\_timer = false;

//////////previousMoveServo7\_timer = currentMillis;

//reset the previousMoveServo7\_timer automatically only if the train will begin to move.

//There are two situations in which the train will begin to move:

// (a) when users ask to move train or

// (b) when the train is not hindered and it continues to finish the previous movement.

allow2updatePreviousMoveServo7\_timer = false; //lock

//default//allow2updatePreviousMoveServo7\_timer = false; //lock

previousMoveServo7\_timer = currentMillis; //update

//Serial.println(F("previousMoveServo7\_timer"));

//Serial.println(previousMoveServo7\_timer);

//synchronize

// capture the latest value of currentMillis

// this is equivalent to noting the time from a clock

// use the same time for all servo 7 movements to keep them synchronized

robotController.moveServo(7, 515, 1000); //set speed

isRunningTrain = true;

//Serial.println(F("durationMoveTrain\_accumulationActual"));

//Serial.println(currentMillis - previousMoveServo7\_timer);

//Serial.println(durationMoveTrain);

} else {//!allow2updatePreviousMoveServo7\_timer

unsigned long durationMoveTrain\_accumulationActual = abs2(currentMillis - previousMoveServo7\_timer); //abs3() can only be used for integer

if (durationMoveTrain\_accumulationActual >= durationMoveTrain) {// only no sensor signal case, we guess

stopTrain();

//it could be wrong// previousDistanceSensor1 = sensor1;

} else {//durationMoveTrain\_accumulationActual < durationMoveTrain

//skip

//do nothing if we have already stopped the train

//continue to move if the train is running

}

}

//update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

if (previousDistanceSensor1 < sensor1) {

previousDistanceSensor1 = sensor1;

} else {//previousDistanceSensor1 >= sensor1

//skip

}

} else {//durationMoveTrain > durationMoveTrain\_Max

//skip //do nothing

}

} else {

//skip//do nothing

}

} else {//set up

//protection:

//moveForward(durationMoveTrain\_Max); allowed max distance //moveForward(durationMoveTrain\_referPoint2jointPoint); in the setup() which means sensor1 310 joint point

//previousDistanceSensor1 = sensor1; //update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

//sensor1 sensor1\_Min allowed min distance

//reference point: 145 (143 - 147 is fine)

if (durationMoveTrain <= durationMoveTrain\_Max) {

robotController.moveServo(7, 515, 1000); //set speed

/\*

//update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

if (previousDistanceSensor1 < sensor1) {

previousDistanceSensor1 = sensor1;

} else {//previousDistanceSensor1 >= sensor1

//skip

}

\*/

isRunningTrain = true;

delay(durationMoveTrain);

stopTrain();

//assume the sensor1 laser beam is not hindered at the very beginnings

//it could be wrong// previousDistanceSensor1 = sensor1;

} else {//durationMoveTrain > durationMoveTrain\_Max

//skip //do nothing

}

}

}

//========================================================================================================================================================================

void stopTrain() {

//default//allow2updatePreviousMoveServo7\_timer = false; //lock

robotController.moveServo(7, 500, 1); //train stop:

//previousDistanceSensor1 = sensor1; //update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

//no update it because it could be not correct

isRunningTrain = false;

//only the train is really stopped for a reasonable reason, not pause not skip, just correct to stop

//In other words, the train finished the request movement, then

isFirstTime2SetDurationMoveTrain\_remainingPlanned = true;

}

//========================================================================================================================================================================

void pauseTrain() {

//default//allow2updatePreviousMoveServo7\_timer = false; //lock

robotController.moveServo(7, 500, 1); //train stop:

//previousDistanceSensor1 = sensor1; //update the previousDistanceSensor1 only if the sensor1 is working well and is not hindered

//no update it because it could be not correct

isRunningTrain = true;

//because it is not a right stop

//besides, if isRunning Train = 0, group action will continue to be run wrongly.

//only the train is really stopped for a reasonable reason, not pause not skip, just correct to stop

//In other words, the train finished the request movement, then

//isFirstTime2SetDurationMoveTrain\_remainingPlanned = true;

//otherwise, default is false during the whole runTime

}

//========================================================================================================================================================================

void moveForwardSafe() {

static unsigned long previousButtonMillis\_timer; //unsigned long

static byte step;

//Serial.println(F("moveForwardSafe().step"));

//Serial.println(step);

if (previousButtonMillis\_timer <= abs2(currentMillis - (long)trainDuration)) {

previousButtonMillis\_timer += trainDuration;

if (step == 0) {

allow2updatePreviousMoveServo7\_timer = true; //unlock

isRunningTrain = true;

step = 1;

} else {

if (step == 1) {

if (0) {// (isHinderedSensor1()) {

//2. represent the remainning time period when there is a barrier.

durationMoveTrain = abs3(durationMoveTrain\_remainingPlanned - abs2(millis() - previousMillis)); //?0.5 delay

//durationMoveTrain\_accumulationActual = abs2(currentMillis - previousMoveServo7\_timer);

//abs3() only applied to the integer

//Serial.println(F("durationMoveTrain In barrierd branch"));

//Serial.println(durationMoveTrain);

allow2updatePreviousMoveServo7\_timer = true; //unlock

step = 2;

isFirstTime2SetDurationMoveTrain\_remainingPlanned = true;

//pause

pauseTrain();

//it could be wrong// previousDistanceSensor1 = sensor1;

//assume the train is moving, just have to wait for pet leaving there.

//there is barrier, the sensor1 value is wrong

//previousDistanceSensor1 = sensor1;

//update it only if distance is correct which means no barrier

} else {//!isHinderedSensor1()

if (isFirstTime2SetDurationMoveTrain\_remainingPlanned) {

isFirstTime2SetDurationMoveTrain\_remainingPlanned = false;

//1.only (once) at the beginning of train movement (before recover from the hindered state),

//2.only (once) at the beginning of train movement (user set),

//the durationMoveTrain\_remainingPlanned will be reset

//only the train is really stopped for a reasonable reason, not pause not skip, just correct to stop

//In other words, the train finished the request movement, then

//isFirstTime2SetDurationMoveTrain\_remainingPlanned = true;

durationMoveTrain\_remainingPlanned = durationMoveTrain;

} else {

durationMoveTrain\_remainingPlanned = abs3(durationMoveTrain\_remainingPlanned - abs2(millis() - previousMillis));

//the value above which is calculated since the very second time. In other words,

//this variable will be updated (all the time) during the required movement

//abs3() can only be used for integer

}

previousMillis = millis();

//Serial.println(F("durationMoveTrain\_remainingPlanned"));

//Serial.println(durationMoveTrain\_remainingPlanned);

//Serial.println(F("durationMoveTrain"));

//Serial.println(durationMoveTrain);

moveForward(durationMoveTrain); //initTrain(sensor1);

}

if (!isRunningTrain) {

step = 0;

}

} else {

if (step == 2) {

//skip

//wait here

//pause

pauseTrain();

//it could be wrong// previousDistanceSensor1 = sensor1;

//assume the train is moving, just have to wait for pet leaving there.

//there is barrier, the sensor1 value is wrong

//previousDistanceSensor1 = sensor1;

//update it only if distance is correct which means no barrier

if (!isHinderedSensor1()) {

step = 1;

}

} else {//step >= 3

//skip

//There is a bug if jump into this hole

}

}

}

}

}

//========================================================================================================================================================================

void moveBackwardSafe() {

static unsigned long previousButtonMillis\_timer; //unsigned long

static byte step;

//Serial.println(F("moveBackwardSafe().step"));

//Serial.println(step);

if (previousButtonMillis\_timer <= abs2(currentMillis - (long)trainDuration)) {

previousButtonMillis\_timer += trainDuration;

if (step == 0) {

allow2updatePreviousMoveServo7\_timer = true; //unlock

isRunningTrain = true;

step = 1;

} else {

if (step == 1) {

if (0) {// (isHinderedSensor1()) {

//2. represent the remainning time period when there is a barrier.

durationMoveTrain = abs3(durationMoveTrain\_remainingPlanned - abs2(millis() - previousMillis)); //?0.5 delay

//durationMoveTrain\_accumulationActual = abs2(currentMillis - previousMoveServo7\_timer);

//abs3() only applied to the integer

allow2updatePreviousMoveServo7\_timer = true; //unlock

step = 2;

isFirstTime2SetDurationMoveTrain\_remainingPlanned = true;

//pause

pauseTrain();

//it could be wrong// previousDistanceSensor1 = sensor1;

//assume the train is moving, just have to wait for pet leaving there.

//there is barrier, the sensor1 value is wrong

//previousDistanceSensor1 = sensor1;

//update it only if distance is correct which means no barrier

} else {//!isHinderedSensor1()

if (isFirstTime2SetDurationMoveTrain\_remainingPlanned) {

isFirstTime2SetDurationMoveTrain\_remainingPlanned = false;

//1.only (once) at the beginning of train movement (before recover from the hindered state),

//2.only (once) at the beginning of train movement (user set),

//the durationMoveTrain\_remainingPlanned will reset

//only the train is really stopped for a reasonable reason, not pause not skip, just correct to stop

//In other words, the train finished the request movement, then

//isFirstTime2SetDurationMoveTrain\_remainingPlanned = true;

durationMoveTrain\_remainingPlanned = durationMoveTrain;

} else {

durationMoveTrain\_remainingPlanned = abs3(durationMoveTrain\_remainingPlanned - abs2(millis() - previousMillis));

//the value above which is calculated since the very second time. In other words,

//this variable will be updated (all the time) during the required movement

//abs3() can only be used for integer

}

previousMillis = millis();

moveBackward(durationMoveTrain); //initTrain(sensor1);

}

if (!isRunningTrain) {

step = 0;

}

} else {

if (step == 2) {

//wait here

//pause

pauseTrain();

//it could be wrong// previousDistanceSensor1 = sensor1;

//assume the train is moving, just have to wait for pet leaving there.

//there is barrier, the sensor1 value is wrong

//previousDistanceSensor1 = sensor1;

//update it only if distance is correct which means no barrier

if (!isHinderedSensor1()) {

step = 1;

}

} else {//step >= 3

//skip

//There is a bug if jump into this hole

}

}

}

}

}