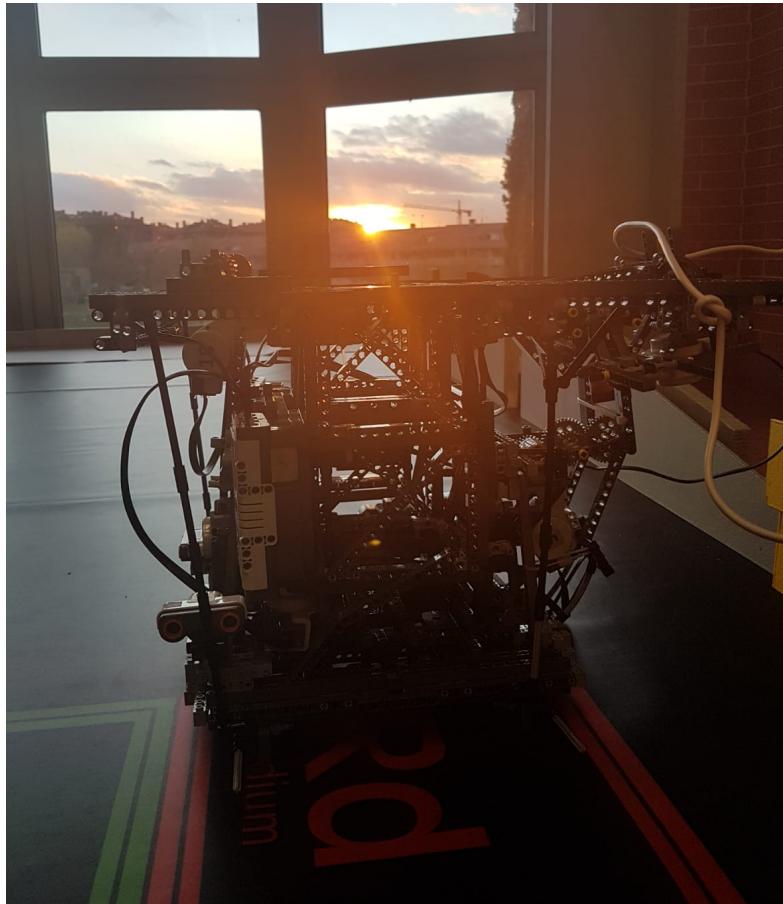


PROYECTO FINAL ROBOTICA PARA TODOS



Realizado por:

- Francisco Javier Adame Cordero
- Francisco Cano Díaz
- Juan Casado Ballesteros
- Juan José Córdoba Zamora
- Imanol García Hernando
- Álvaro Gómez Martínez
- Gabriel López Cuenca
- Álvaro Vaya Arboledas

Contenido

1.	OBJETIVOS A CUMPLIR.....	2
2.	OBJETIVOS CUMPLIDOS	3
3.	HARDWARE	5
1.	ESTRUCTURA	5
2.	PINZA Y BRAZO	5
3.	CLASIFICADOR	6
4.	SENSORES DE DISTANCIA.....	7
5.	EXPERIMENTO	8
6.	ESTRUCTURA FINAL	9
4.	SOFTWARE	10
1.	ODOMETRÍA	11
2.	ULTRASONIDOS	15
3.	BLUETOOTH.....	19
4.	CLASIFICADOR/SENSOR DE COLOR.....	20
5.	SERVO	21
5.	ORGANIZACIÓN.....	24
6.	CRÍTICA.....	25

1. OBJETIVOS A CUMPLIR

En este apartado vamos a desarrollar los objetivos que nosotros considerábamos más importante a la hora de realizar y programar el robot. Los objetivos por cumplir se encuentran influenciados por la normativa de Eurobot 2019 que puede consultarse [aquí](#). Los objetivos que nos propusimos son los siguientes:

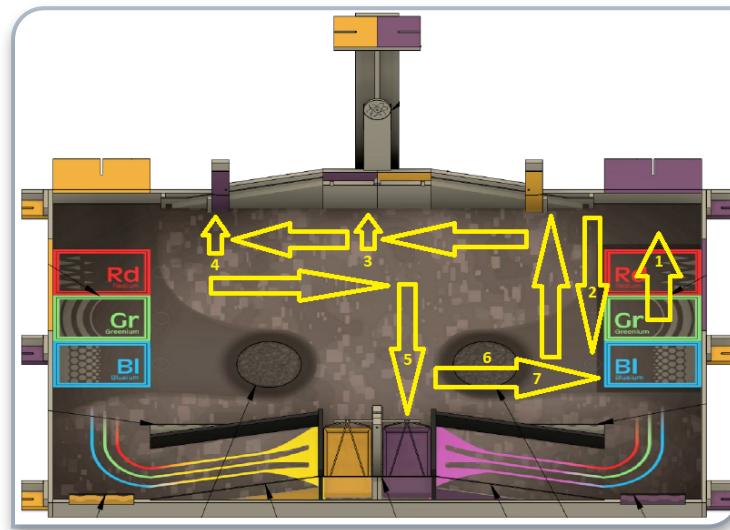
- Conseguir que el robot se moviera completamente recto y los giros los hiciese con muy poco error.
- Conseguir que el robot siguiera la ruta que nosotros queríamos.
- Realización de un experimento funcional que consiguiese subir un electrón a lo alto.
- Activación del experimento.
- Realización de un brazo con una pinza totalmente funcional.
- Utilizar el acelerador de partículas para conseguir que se descubriese el Goldenium y poder cogerlo para posteriormente llevarlo a la báscula y soltarlo.
- Conseguir que el brazo cogiese los diferentes átomos Redium, Greenium y Blueium (tanto los que estaban en el suelo como los que estaban en las paredes) para clasificarlos en los dispensadores del robot para después llevarlas a un lado o a otro en función de su peso/color (Las piezas más pesadas llevarlas a la báscula y luego clasificar las piezas por su color en la tabla periódica).
- Utilizar bluetooth para poder unir varios ladrillos.
- Conseguir trabajar en equipo (proponer ideas, visualizar los problemas antes de que ocurriesen, solucionar los problemas que iban surgiendo) para poder sacar el trabajo adelante.
- Trabajar y familiarizarnos con las metodologías ágiles, concretamente con scrum. Realizar reuniones diarias (cada vez que íbamos al laboratorio para trabajar) de cómo íbamos y que íbamos a hacer ese día.

2. OBJETIVOS CUMPLIDOS

Los objetivos que nos propusimos al inicio de la práctica fueron muy extensos y casi imposible de realizarlos todos, por eso tuvimos que elegir entre los objetivos que nosotros creímos más importantes, nos llevaban menos tiempo y los que nos iban a dar más puntos.

Los objetivos que finalmente cumplimos fueron los siguientes:

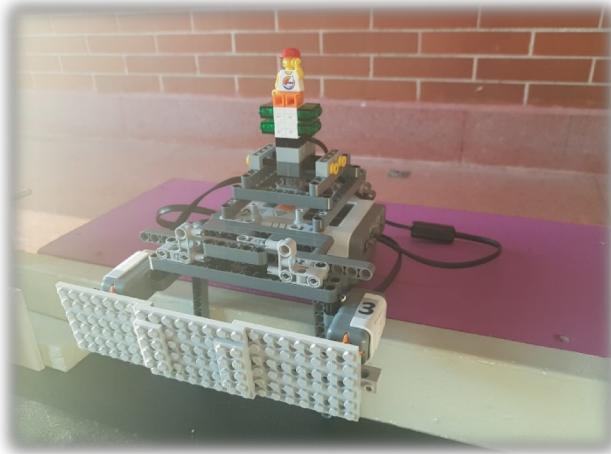
1. Conseguir que el robot se moviera y realizase la ruta que queríamos. Foto del recorrido:



<https://youtu.be/eDtIjeve1g4>

https://youtu.be/_kXEVo65mwE

2. Activación del experimento: realización de un experimento funcional que consiguiere subir un electrón a lo alto.



<https://youtu.be/w5wV5yRSfOs>

3. Realización de un brazo con una pinza totalmente funcional (al no tener servos tuvimos que programar los motores para que funcionasen como servos).

<https://youtu.be/9qLxXKEjcvA>

4. Arrastrar los átomos que estaban en el suelo para llevarlos a la tabla periódica.

<https://youtu.be/auBwFaBb3n0>

3. HARDWARE

Desde un principio nos planteamos un proyecto grande y ambicioso de modo que uno de los principales retos a superar sería crear una estructura capaz de soportar cualquier tipo de sensor u actuador que deseáramos introducir.

Otra característica de la que quisimos dotar a la construcción del robot es la de que fuera completamente modular de modo que pudiéramos acoplar o retirar componentes sin que eso afectara al conjunto del robot.

Se aprecian los siguientes componentes independientes cada uno separable del resto y plenamente funcional estando solo.

1. ESTRUCTURA

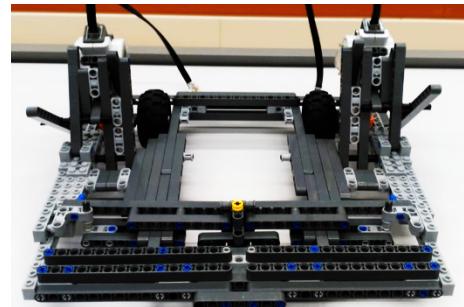
Comenzamos la construcción de la estructura creando un frame que fuera suficientemente rígido y resistente como para no hundirse aplicando presión con la mano.

Esta es la primera versión de este que posteriormente tuvimos que modificar retirando la rueda loca de la parte central para sustituirla por dos L de Lego, una en cada esquina de la plataforma de modo que tocaran el suelo. Con estas L logramos que la plataforma fuera más estable que ahora teníamos un total de 4 puntos de apoyo. Adicionalmente las L a diferencia de la rueda loca nos permitían subir por la rampa ya que elevaban ligeramente al robot del suelo.

Decidimos integrar los motores dentro de la base ya que debido a que íbamos a utilizar odometría era vital que estuvieran correctamente sujetos y distribuyendo entre ellos la misma cantidad de peso del robot para favorecer a que no girara uno más que el otro.

La estructura tiene dos funciones principales dentro del robot:

- Sirve de soporte para todo el resto de las componentes de modo que se busca que estos no se unan en ningún momento entre si no que se añadan a la estructura para favorecer el modularidad del hardware.
- Nos permite empujar las piezas del suelo. La estructura puede golpear las piezas en su centro y con dos barras salientes en forma de U puede empujarlas sin peligro de que se escapen por los laterales al moverse o girar.

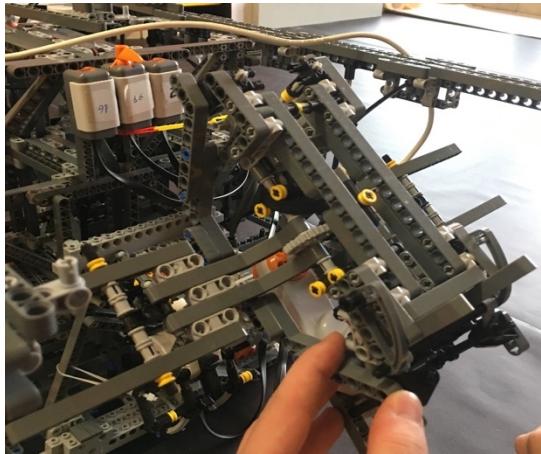


2. PINZA Y BRAZO

Dotamos al robot de este actuador de modo que pudiera tomar piezas de las paredes. Tanto la pinza como el brazo funcionan como servos que mantienen una posición deseada haciendo fuerza para quedarse en ella.

Para aumentar la fuerza y reducir la velocidad de los movimientos dotamos a la pinza de una reductora a través de un tornillo sin fin y al brazo de otra reductora a través de un sistema de engranajes. No obstante, en el caso del brazo esto no fue suficiente para permitirle levantar las piezas más pesadas de modo que incorporamos una goma elástica para aprovechar la ley de hook de modo que la fuerza a ejercer para elevar las piezas se redujera.

El diseño final de la pinza se modificó respecto al diseño en la foto de modo que en vez de ser una única barra la que formara la zona de agarre fueran dos, de modo que una quedara por encima del centro de la pieza al cogerla y otra por debajo quedando así estas sujetas de un modo mucho más firme.



3. CLASIFICADOR

Tras leer la documentación decidimos que la mejor forma de resolver el problema planteado sería clasificar los discos situados en las paredes de modo que pudiéramos obtener la mayor cantidad de puntos posibles de ellos. Ubicaríamos los 6 discos más pesados en una de las torres, (se muestran en fase de construcción más abajo) mientras que el resto de los discos irían a la otra torre.

Los discos más ligeros serían depositados en su color ya que para poderlos clasificar habremos leído su color con el sensor del color y el bumper que se ven en la primera foto, guardamos estos colores leídos en un array para recordarlos al irlos a sacar.

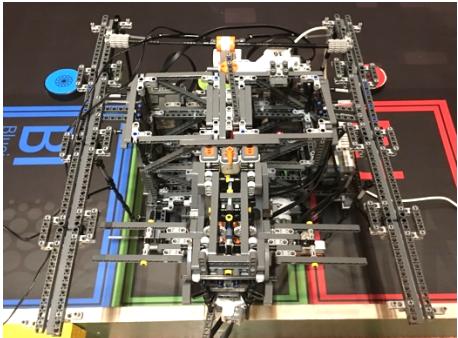
Para sacarlos de las torres utilizamos dos motores que actúan como dispensadores en la parte inferior de estas.



Sensores de color.



Dispensadores.

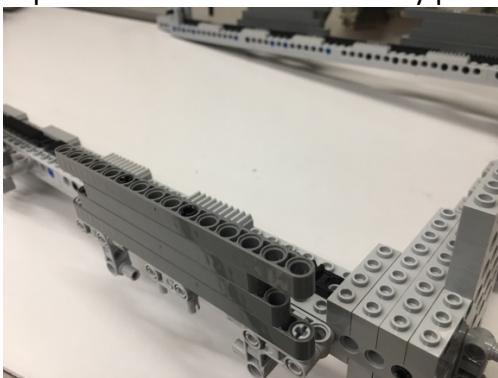


Clasificación.



Torres.

El proceso total de la clasificación hace uso de un elemento adicional que empuja las piezas desde la pinza hasta el lector de color y posteriormente hasta las torres utilizando estos patines.

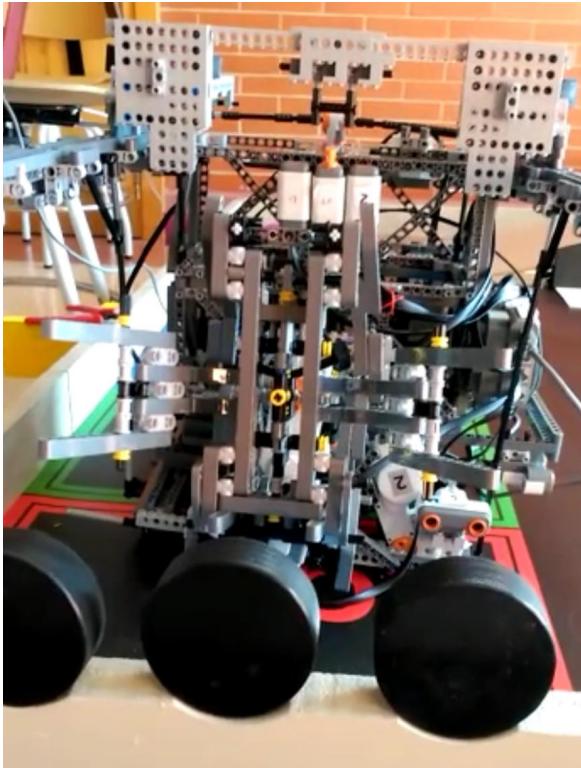


Los pasos a seguir para clasificar una pieza son los siguientes:

1. Bajamos el brazo hasta situarlo a la altura de las piezas en las paredes.
2. Cerramos la pinza hasta agarrar la pieza.
3. Subimos el brazo hasta dejarlo vertical.
4. Abrimos la pinza hasta la mitad de modo que la pieza pueda caer sobre los carriles pero al volver a bajar el brazo la anchura de la pinza semi-abierta pueda pasar entre los raíles.
5. Bajamos en brazo a la mitad para que los raíles puedan moverse por encima de él.
6. Extendemos el rail.
7. Terminamos de abrir la pinza.
8. Subimos del todo la pinza hasta la vertical.
9. Replegamos el rail empujando la pieza hasta que esta toque el bumper.
10. Cuando lluegue al bumper leemos el color de la pieza.
11. Ubicamos el motor del clasificador al lateral correspondiente para que pueda empujar la pieza a la torreta correspondiente.
12. Terminamos de replegar los raíles.
13. Empujamos la pieza a su torreta.
14. Posteriormente con los dispensadores sacamos la pieza de la torreta.

4. SENORES DE DISTANCIA

Para complementar a la odometría pensamos que sería una buena idea utilizar cuatro sensores de distancia para ubicarnos con más precisión dentro del campo. Dichos sensores se colocarían apuntando de forma perpendicular a cada uno de las cuatro laterales del robot.

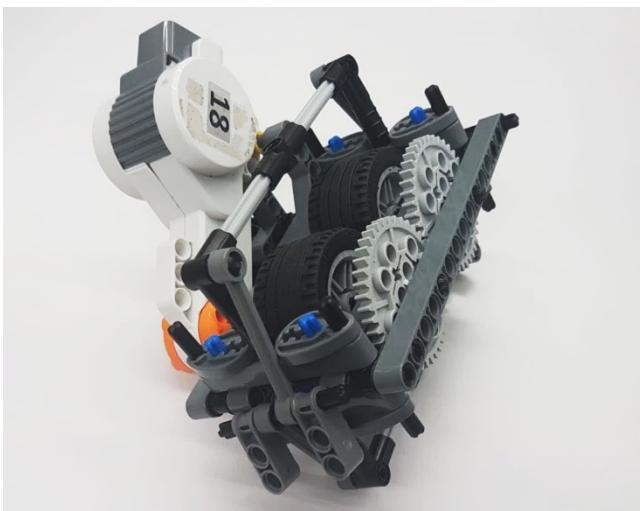


En esta foto se puede ver la forma final de la pinza con el nuevo agarre doble.
En la parte inferior derecha se ve uno de los sensores de distancia por ultrasonidos.
También se ve la barrera que desliza sobre los raíles en la parte superior de la imagen.
En su centro se ve el diseño final de las torres.

5. EXPERIMENTO

Para realizar el experimento utilizamos un ladrillo con dos bumper de modo que se pulsara el que se pulsara este se activara. Al activar el experimento comenzaría a sonar la música de Piratas de Caribe en 8bits, además se encendería un LED.

Para subir el electrón colocamos una cuerda por la que este podría subir con la ayuda de un único motor que estaría moviendo dos ruedas. Tuvimos que alargar un cable de Lego para que pudiera alimentar el motor hasta que este llegara arriba.



Electrón.

6. ESTRUCTURA FINAL

Aquí se puede ver la estructura final del robot donde se pueden apreciar todos sus componentes principales, la pinza con el brazo, las torres clasificadoras y los railes deslizantes.



A nivel de hardware logramos cumplir todos los requisitos planteados, principalmente gracias al prototipado rápido que Lego permite el cual va muy en línea con la metodología aplicada que recordemos que ha sido SCRUM, la cual es una metodología ágil.

Estructura final.

4. SOFTWARE

En este apartado se explicará el software realizado para la práctica final incluyendo tanto las partes que fueron presentadas en la prueba final y las partes que fueron diseñadas, pero no se pudieron presentar en la prueba debido a problemas de última hora o porque se decidió no implementarlo finalmente debido a que no aportaban ninguna funcionalidad final.

Se describirá en código final dividido en los siguientes apartados:

1. ODOMETRIA
2. ULTRASONIDOS
3. BLUETOOTH
4. CLASIFICADOR/SENSOR DE COLOR
5. SERVO

1. ODOMETRÍA

En la parte de odometría de nuestro código nos hemos basado en la parte de odometría de empleada en la práctica 4 de seguimiento de una ruta usando odometría para diseñar un algoritmo que por medio de un array de movimientos trazara la ruta deseada.

Para lograr el objetivo de odometría el código se puede dividir en dos partes fundamentales:

- posicion: esta parte del código se encargaba de por medio de odometría obtener la posición actual basándose en la rotación de los motores desde el inicio del programa por medio de la función **MotorRotationCount** los datos obtenidos de esta task se suministran a la task movimiento encargada de mover el robot según la ruta deseada.

```
task posicion()
{
    int d = 0;
    float d_anterior = 0;
    float d_inc = 0;
    int i = 0;
    float i_anterior = 0;
    float i_inc = 0;
    float theta = 0;
    float distance = 0;
    float new_x = 0;
    float new_y = 0;
    float constanteAngulo = (2 * PI * RADIO) / (360*D);
    float constanteDistancia = (2 * PI * RADIO) / (360*2);
    while(true)
    {
        Wait(TIME);
        d = MotorRotationCount(MOTORORDER);
        i = MotorRotationCount(MOTORIZQ);
        d_inc = d - d_anterior;
        i_inc = i - i_anterior;
        d_anterior = d;
        i_anterior = i;
        theta += constanteAngulo * (d_inc - i_inc);
        distance = constanteDistancia * (d_inc + i_inc);
        new_x = cos(theta)*distance;
        new_y = sin(theta)*distance;
        Acquire(semaforo);
        x += new_x;
        y += new_y;
        angle = theta;
        dist += distance;
        Release(semaforo);
    }
}
```

- Movimiento: esta parte del código se encarga de mover el robot siguiendo un array de movimientos y por medio de los datos suministrados por la task de posicionamiento es capaz de reconocer si ha alcanzado la distancia especificada en la posición actual del array de movimientos y pasar a analizar la siguiente posición. Por medio del control PID se regulaba el movimiento del robot considerando la posición actual, obtenida de la task posición, y la distancia de la posición actual del array de movimiento.

```

void move()
{
    int nextMovement = 0;
    int tam = getTam();
    float errorX = 0;
    float error_anteriorX = 0;
    float error_accX = 0;
    float to_doX = 0;
    float errorY = 0;
    float error_anteriorY = 0;
    float error_accY = 0;
    float to_doY = 0;
    int timesInOk = 0;
    float posX = 0;
    float posY = 0;
    float to_doA = 0;
    float to_doB = 0;
    long mA = 0;
    long mB = 0;
    bool action = true;
    unsigned long timer = 0;
    while (action)
    {
        Acquire(semaforo);
        posX = dist * 100;
        posY = angle * 100;
        Release(semaforo);
        errorY = getArrayY(nextMovement) - posY;
        errorX = getArrayX(nextMovement) - posX;
        error_accY = constr(error_accY + errorY, MAX_ACC, -MAX_ACC);
        error_accX = constr(error_accX + errorX, MAX_ACC, -MAX_ACC);
        if(getArrayGiro(nextMovement))
        {
            if(abs(errorY) < MARGIN){
                if(timesInOk < MARGIN){
                    timesInOk++;
                } else {
                    if (nextMovement < tam){
                        nextMovement++;
                        timer=0;
                        error_accX = 0;
                        error_accY = 0;
                    } else {
                        action = false;
                    }
                }
            }
        }
    }
}

```

```

else {
    timesInOk = 0;
}
to_doY = errorY*PG + error_accY * IG + (errorY-error_anteriorY)*DG;
to_doY = constr (to_doY, MAX_GIRO, -MAX_GIRO);
} else {
    if (abs(errorX) < MARGIN){
        if (timesInOk < MARGIN){
            timesInOk++;
        } else {
            if (nextMovement < tam){
                nextMovement++;
                timer = 0;
                error_accX = 0;
                error_accY = 0;
            } else {
                action = false;
            }
            timesInOk = 0;
        }
    }else {
        timesInOk = 0;
    }
    to_doY = errorY*PY + error_accY*IY + (errorY-error_anteriorY) * DY;
}
to_doX = errorX*PX + error_accX*IX + (errorX-error_anteriorX) * DX;
to_doX = constr(to_doX, MAX_GO, -MAX_GO);
to_doA = to_doX + to_doY;
to_doB = to_doX - to_doY;
if (to_doA > 0){
    mA = to_doA;
    OnFwd(MOTORORDER,constraint(mA, MAX_VEL, -MAX_VEL));
} else {
    mA=to_doA;
    OnRev(MOTORORDER, constraint(-mA, MAX_VEL, -MAX_VEL));
}

if (to_doB > 0)
{
    ClearScreen();
    mB = to_doB;
    NumOut(1, LCD_LINE3,to_doB);
    NumOut(1, LCD_LINE4,mB);
    OnFwd(MOTORIZQ, constraint(mB, MAX_VEL, -MAX_VEL));
} else {
    mB = to_doB;
    ClearScreen();
    NumOut(1, LCD_LINE5,to_doB);
    NumOut(1, LCD_LINE6,mB);
    OnRev(MOTORIZQ, constraint(-mB, MAX_VEL, -MAX_VEL));
}

```

```

        error_anteriorY = errorY;
        error_anteriorX = errorX;
        if (timer > MAX_TIMER){
            nextMovement++;
            timer=0;
            if (nextMovement >= tam){
                action = false;
            }
            }else {timer++;}
            Wait(TIME);
        }
        OnFwd(MOTORORDER, 0);
        OnFwd(MOTORIZQ, 0);
        Wait(TIME);
    }
}

```

Los arrays de rutas empleados para lograr el movimiento del robot son los siguientes:

```

// Primer movimiento
int tam1 = 5;
float arrayX1 [tam1] = {100,100,100,50,150};
float arrayY1 [tam1] = {0,0,0,0,0};
bool arrayG1 [tam1] = {0,0,0,0,0};

// Segundo movimiento
int tam2 = 12;
float arrayX2[tam2] = {30,20,0,-15,-15,-55,-55,-60,-15,-15,50,100};
float arrayY2[tam2] = {0,0,-110,-110,-35,-35,0,0,0,222,222,222};
bool arrayG2[tam2] = {0,0,1,0,1,0,1,0,0,1,0,0};

```

2. ULTRASONIDOS

En la parte de ultrasonidos de nuestro código se emplean 4 sensores de ultrasonidos conectados a la base del robot que detectan la distancia a la que se encuentra un obstáculo. Por medio de los ultrasonidos se pretendía apoyar a la odometría del movimiento del robot actualizando los valores de posición por vía bluetooth en el caso de detectarse una desviación en la ruta. Esta parte no se empleó en el desarrollo final del proyecto dado que las paredes del campo no eran lo suficientemente altas para que los sensores los detectaran, pero las partes del código desarrolladas son las siguientes:

- *actualizador_ultrasonido*: esta task se encarga de obtener las distancias a las que se encuentra cada sensor de ultrasonidos y suministrárselas a una task de bluetooth que se encargara de enviarlas.

```

task actualizador_ultrasonido()
{
    SetSensorLowspeed(IN_1);
    SetSensorLowspeed(IN_2);
    SetSensorLowspeed(IN_3);
    SetSensorLowspeed(IN_4);
    float prueba = 0;
    float medidaAcumulada1 = 0;
    float medidaAcumulada2 = 0;
    float medidaAcumulada3 = 0;
    float medidaAcumulada4 = 0;
    int sensorAMirar = 1;
    string str1,str2,str3,str4;
    int i = 0;
    float valor1;
    float valor2;
    float valor3;
    float valor4;

    while(true)
    {

        for(int i = 0; i<medidasTomadas; i++)
        {
            medidaAcumulada1+=SensorUS(IN_1);
            medidaAcumulada2+=SensorUS(IN_2);
            medidaAcumulada3+=SensorUS(IN_3);
            medidaAcumulada4+=SensorUS(IN_4);
        }
        Acquire(semaforo);
        sensores[0] = medidaAcumulada1/medidasTomadas;
        Release(semaforo);
        Acquire(semaforo);
        sensores[1] = medidaAcumulada2/medidasTomadas;
        Release(semaforo);
        Acquire(semaforo);
        sensores[2] = medidaAcumulada3/medidasTomadas;
        Release(semaforo);
        Acquire(semaforo);
        sensores[3] = medidaAcumulada4/medidasTomadas;
        Release(semaforo);
        medidaAcumulada1 = 0;
        medidaAcumulada2 = 0;
        medidaAcumulada3 = 0;
        medidaAcumulada4 = 0;
        Wait(200);
    }
}

```

- Blue: esta task se encarga de recibir las medidas suministradas por el actualizador_ultrasonido y convertirlas a una cadena que se enviara por medio de bluetooth al maestro para que este actualice los datos de ser necesario. Para enviar los datos se empleará la función **StrCat** que enviará las posiciones con el siguiente formato: 030.1010.2100.0001.0 para las posiciones de los sensores 30.1, 10.2, 100.0 y 1.0.

```

string convStr(float v)
{
    string aux = "";
    if (v > 999.9)
    {
        return "999.9";
    }else{
        if (v < 10)
        {
            aux = "00";
        }else if(v < 100)
        {
            aux = "0";
        }
        return StrCat(aux, FormatNum("%.1f",v));
    }
}
task blue(){
    string str1,str2,str3,str4;
    string envio;
    float valor1;
    float valor2;
    float valor3;
    float valor4;
    while(true){

        Acquire(semaforo);
        valor1 = sensores[0];
        valor2 = sensores[1];
        valor3 = sensores[2];
        valor4 = sensores[3];
        Release(semaforo);
        envio = StrCat(convStr(valor1), convStr(valor2), convStr(valor3), convStr(valor4));
        SendRemoteString(BT_CONN,OUTBOX, envio)
        Wait(200);
    }
}

```

Por la otra parte podemos encontrar en el maestro una función que se encargara de recibir la cadena enviada y descomponerla en floats para que puedan ser analizados por odometría.

```
task main(){
    float in1, in2, in3, in4;
    BTCheck(BT_CONN);
    while(true){
        ReceiveRemoteString(INBOX, true, output);
        in1 = StrToNum(SubStr(output, 0, 5));
        in2 = StrToNum(SubStr(output, 5, 5));
        in3 = StrToNum(SubStr(output, 10, 5));
        in4 = StrToNum(SubStr(output, 15, 5));
        Wait(500);
        ClearScreen();
    }
}
```

3. BLUETOOTH

En la parte de bluetooth lo organizamos con 2 ladrillos, uno que era el maestro quien era el que tenía el programa principal (odometría y movimiento), también es quién manda una señal al esclavo, el cual tenía conectado los motores de la pinza, el brazo y cargado el programa que los movía (está constantemente esperando a que le llegue una señal para poder subir el brazo, bajar el brazo, abrir la pinza o cerrar la pinza).

```
sub BTCheck(int conn){  
    CommBTConnectionType args;  
    args.Name = "NXT18"; // whatever the slave NXT's name is  
    args.ConnectionSlot = conn; // this is the desired connection slot (the above code uses 1)  
    args.Action = TRUE; // could use some #define with a non-zero value to connect. 0 == disconnect  
    if(!BluetoothStatus(conn)==NO_ERR)  
    {  
        SysCommBTConnection(args); // try to connect.  
    }  
}
```

```
int recibir(int bt_conn){  
    int in;  
    BTCheck(bt_conn); //check master connection  
    ReceiveRemoteNumber(INBOX,true,in);  
    return in;  
}
```

```
void mandar(int msg, int bt_conn){  
    BTCheck(bt_conn);  
    SendRemoteNumber(bt_conn, OUTBOX, msg);  
}
```

4. CLASIFICADOR/SENSOR DE COLOR

En la parte del clasificador/sensor de nuestro código se emplean 2 bampers y 1 sensor de luz para la detección del color de la ficha que se deposita sobre el sensor, para ello se espera a que el bumper se active y entonces el sensor de luz toma la medida, esta medida se convierte por medio de condicionales y márgenes de luz a un color que será suministrado a una función que dependiendo del motor mueve el clasificador para dejar caer la ficha en una torre u en otra. Esta funcionalidad no se llegó a mostrar en la práctica final debido a que por problemas de bluetooth y tiempo no se pudo llegar a dar el caso en el que ficha fuera subida al sensor. El código se divide en las siguientes partes:

- getColor: esta task se encarga de obtener el color de la ficha cuando los bampers detectan presión convirtiendo la medida tomada por el sensor de luz a una letra R, G, B, X dependiendo del color, esta letra se le suministra a la función encargada de mover el clasificador.

```
string color(){
    int luz1=0;
    int luz2=0;
    float media=0;
    bool datoLeido=false;
    string color;
    for (int i=0; i<8; i++){
        luz1 += Sensor(IN_1);
        luz2 += Sensor(IN_2);
        Wait(50);
    }
    media = ((luz1/8) + (luz2/8))/2;
    if ((63 < media) && (media < 75)){
        color = "R" ;
    } else if ((50 < media) && (media < 62)){
        color = "G" ;
    } else if ((40 < media) && (media < 49)){
        color = "B" ;
    } else {
        color="X" ;
    }
    return color;
}

task getColor() {
    string col;
    SetSensor(IN_4,SENSOR_TOUCH);
    SetSensorLight(IN_1);
    SetSensorLight(IN_2);
    while(true){
        until(SENSOR_4==0);
        until (SENSOR_4==1);
        col = color();
        TextOut(50,0,col);
    }
}
```

- clasificar: esta función se ejecuta una vez se detecta un color y mueve el motor del clasificador hacia la posición contraria a la que debe empujar la ficha, espera a que los sensores se desactiven al avanzar la ficha y se vuelve a mover para empujar la ficha hacia la torre correcta.

5. SERVO

En la parte del servo de nuestro código se pueden distinguir 2 partes del código una primera parte encargada de obtener la posición actual de rotación de los motores y la segunda parte encargada de mover los motores a las posiciones deseadas. Este código se encontraba implementado, pero no se pudo mostrar en la demostración final debido a que era llamado por bluetooth y había problemas de conexión. Las partes de este código se pueden dividir en las siguientes:

- posicion: esta tarea se encarga de obtener la posición actual de los motores y suministrársela a la función encargada de moverlos. Los motores detectados son los de la pinza, el brazo y la barrena.

```
task posicion()
{
    while(true)
    {
        Acquire(semaforoBrazo);
        anguloActualBrazo = MotorRotationCount(MOTORBRAZO);
        Release(semaforoBrazo);
        Acquire(semaforoPinza);
        anguloActualPinza = MotorRotationCount(MOTORPINZA);
        Release(semaforoPinza);
        Acquire(semaforoBamper);
        anguloActualBamper = MotorRotationCount(MOTORBAMPER);
        Release(semaforoBamper);
    }
}
```

- movimiento: esta tarea se encarga de por medio de control PID mover los motores a las posiciones de destino deseadas comparando la posición de destino con la posición actual para realizar movimiento mediante el control PID. Las posiciones de destino son variables globales que son establecidas de manera externa a esta función.

```

task movimiento()
{
    long errorBrazo = 0, errorPinza= 0, errorBamper = 0;
    float integralBrazo = 0, integralPinza = 0, integralBamper = 0;
    float constantesBrazo = 0, constantesPinza = 0, constantesBamper = 0;
    int countB = 0;
    int velocidadBrazo, velocidadPinza, velocidadBamper;
    while(true)
    {
        Acquire(semaforoBrazo);
        errorBrazo = anguloBrazo - anguloActualBrazo ;
        if(checkAngulo(anguloBrazo, anguloActualBrazo, MARGENBRAZO)){
        }else{
            integralBrazo = constraint(integralBrazo, MAX_INTEGRAL_BRAZO_BAJAR, -
MAX_INTEGRAL_BRAZO_BAJAR) + errorBrazo;
        } // setting brazokp or ki when setting the angle
        constantesBrazo = getbrazokp(anguloBrazo, anguloActualBrazo)*errorBrazo +
getbrazoki(anguloBrazo, anguloActualBrazo)*integralBrazo;
        Release(semaforoBrazo);
        velocidadBrazo = constantesBrazo;
        Acquire(semaforoPinza);
        errorPinza = anguloPinza - anguloActualPinza ;
        timerC++;
        //integralPinza = integralPinza + errorPinza;
        if(checkAngulo(anguloBrazo, anguloActualBrazo, MARGENBRAZO)){
        }else{
            integralBrazo = constraint(integralBrazo, MAX_INTEGRAL_BRAZO_BAJAR, -
MAX_INTEGRAL_BRAZO_BAJAR) + errorBrazo;
        }
        if (timerC > LIMITE)
        {
            setAnguloPinza(anguloActualPinza);
        } else{
            timerC++;
        }
        constantesPinza = getpinzakp(anguloBrazo, anguloActualBrazo)*errorPinza +
getbrazoki(anguloBrazo, anguloActualBrazo)*integralPinza;
        Release(semaforoPinza);
        velocidadPinza = constantesPinza;
        // Calculo velocidad motor Bamper
        Acquire(semaforoBamper);
        errorBamper = anguloBamper - anguloActualBamper ;

        integralBamper = integralBamper + errorBamper;
        constantesBamper = bamperkp*errorBamper + bamperki*integralBamper;
        Release(semaforoBamper);
        velocidadBamper = constantesBamper;
        OnFwd(MOTORBRAZO, constraint(velocidadBrazo, MAX_VEL, -MAX_VEL));
        OnFwd(MOTORPINZA, constraint(velocidadPinza, MAX_VEL, -MAX_VEL));
        OnFwd(MOTORBAMPER, constraint(velocidadBamper, MAX_VELBAMPER, -
MAX_VELBAMPER));
        Wait(TIMER);
    }
}

```

Adicionalmente este apartado tiene una parte de bluetooth que recibe del maestro un número que indica que acción realizar. El código sería el siguiente:

```
void ejecutar(int mensage)
{
    switch(mensaje)
    {
        case 0:
            bajarBrazo();
            break;
        case 1:
            subirBrazo();
            break;
        case 2:
            cerrarPinza();
            break;
        case 3:
            abrirPinza();
            break;
    }
}
task rutina()
{
    while(true)
    {
        ejecutar(recibir(BT_CONN));
    }
}
```

5. ORGANIZACIÓN

La organización del equipo la hemos hecho en base a la metodología aplicada (SCRUM) de modo que todo lo realizado lo hemos hecho en base a pequeñas iteraciones coordinadas por un SCRUM MÁSTER.

Para realizar la organización hemos intentado cumplir dos objetivos:

- Nadie debe estar trabajando solo, como mínimo habrá dos personas realizando cada tarea.
- Todo el mundo debe tener la tarea siguiente a la que iba a realizar definida antes de que termine aquella que está realizando. Las posibilidades de asignación de tareas futuras serían comenzar una nueva o ayudar a una pareja que fuera atrasada en la tarea que estaba realizando.
- En la medida en que sea posible a cada persona se le asignará la tarea que más le guste sea hardware o software su preferencia.

Al inicio de cada día realizamos una reunión de control para que todos supiéramos el estado del proyecto y de cada una de sus partes. El primer día hubo una reunión mucho más larga en la que definimos los objetivos que queríamos realizar, la forma del proyecto final y algunos objetivos de la organización, así como quien sería el SCRUM MÁSTER.

Para compartir los vídeos, fotos o código hemos utilizado WhatsApp.

6. CRÍTICA

Al ser tantas personas para realizar el trabajo pensamos que podríamos hacer un proyecto mucho más grande del que finalmente se ha podido realizar debido al tiempo que se ha tenido.

Para otro proyecto similar tendremos en cuenta los siguientes fallos observados en la realización de este.

- Para evitar crear un proyecto que luego termine quedando inacabado o sin toda la funcionalidad planeada realizada en futuros proyectos hemos observado que es necesario trabajar de forma iterativa. En vez de intentar atacar todo el problema de una sola vez es recomendable dividir este en pequeños objetivos que se puedan ordenar por prioridad y que se vayan resolviendo poco a poco en función de ella.
- A la hora de compartir ciertos tipos de archivos como fotos o vídeos WhatsApp es una buena herramienta, pero para compartir código u otros archivos de texto deja bastante que desear. En futuros proyectos utilizaremos desde un inicio un gestor de repositorios como git para compartir los archivos.
- Algunas de las reglas de la EUROBOT 2019 no las hemos cumplido, principalmente las referentes a los tamaños. Esto se debió a que al incorporar alguna de las partes (los raíles principalmente) los cuales habían sido menos pensados en la fase de diseño.
- Nuestro proyecto relegaba gran parte de su funcionalidad en el bluetooth, parte que al final nos dio muchos fallos y nos impidió completar el proyecto a tiempo. Para futuros proyectos evitaremos relegar tan fuerte mente en una única tecnología sin tener como mínimo otras posibles alternativas pensadas para reemplazarla en caso de que no la logremos implementar.