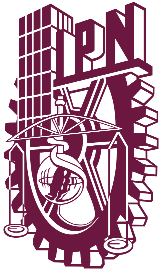
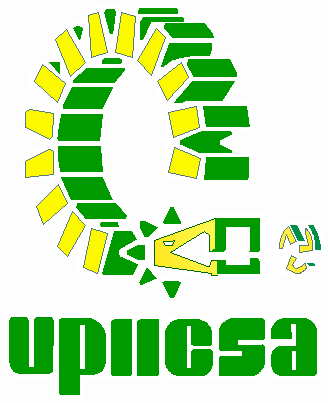
Marco 7



**INSTITUTO POLITECNICO NACIONAL**

UNIDAD PROFESIONAL INTERDISCIPLINARIA DE

INGENIERÍA Y CIENCIAS SOCIALES Y ADMINISTRATIVAS

------------------------------------------------------------------------------------------

**Tripod**

------------------------------------------------------------------------------------------



|  |  |
| --- | --- |
| **NOMBRE:** | * Alamar Martínez Javier Alejandro * Pérez Chamorro Noe Isaí * Ruíz Hernández Pamela Guadalupe * Salvador Lugo Sergio |
| **SECUENCIA:** | 2NM50 |
| **UNIDAD DE APRENDIZAJE:** | Métodos Numéricos |
| **PROFESOR:** | Ríos Suriano Francisco Javier |



Durante el semestre del periodo 18-1 en la materia Métodos Numéricos el equipo usará el equipo “Lego Mindstorms NXT” para desarrollar el proyecto final de la materia.

El cual consta en realizar el armado y programaciónn de un trípode controlado guiado autónomamente por un sensor de ultra sonido por un dispositivo móvil.

Se entregará

1er DPTAL;

Se comprará y pesará el material a utilizar

Se sacarán las fórmulas de los motores y se verá que aguante el peso.

2do DPTAL

Armado el modelo del Tripod

Pruebas de voltaje, es necesario saber cuánto voltaje necesitará el Tripod en cuestión

3er DPTAL.

Realización de programa para el funcionamiento y comportamiento del Tripod

# Introducción

La realización de proyectos de robótica otorga al estudiante la capacidad de aprender importantes conocimientos a través de la construcción, programación y prueba de los robots. Durante este proceso el estudiante se encontrará con conceptos claves que se relacionan con las ciencias de la computaciónn, matemáticas aplicadas, ciencias en general, trabajo en equipo y comunicación. Al mismo tiempo el estudiante aprende el proceso de explorar, planificar y resolver problemas. También se familiarizar con el principio de dividir un proyecto en pequeñas partes y así lograr una solución metódica y más abordable.

Este automóvil de carreras está diseñado para verse y dirigirse como un automóvil real, con dirección giratoria en las ruedas delanteras. También está diseñado para la velocidad, con engranajes para aumentar la velocidad de las ruedas motrices traseras.

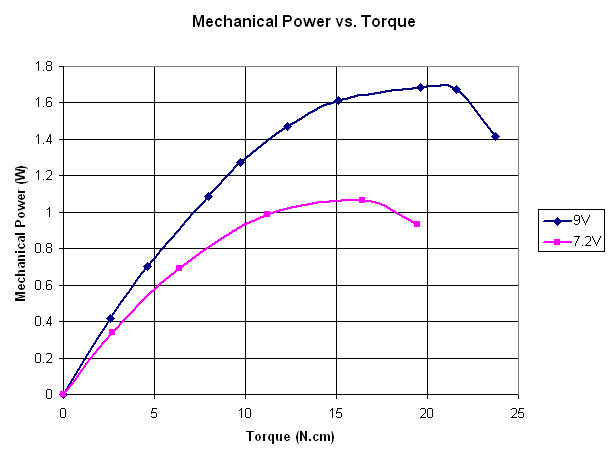
**Características del motor NXT**

[La página de comparación de motores de 9V](http://www.philohome.com/motors/motorcomp.htm) muestra algunas características de este motor considerado como un motor de 9V, pero aquí hay más información sobre su uso con un NXT (¡su uso principal!)

Las curvas a continuación muestran la velocidad de rotación del motor NXT (Rotaciones por minuto) frente al nivel de potencia del motor (ciclo de trabajo de suministro).

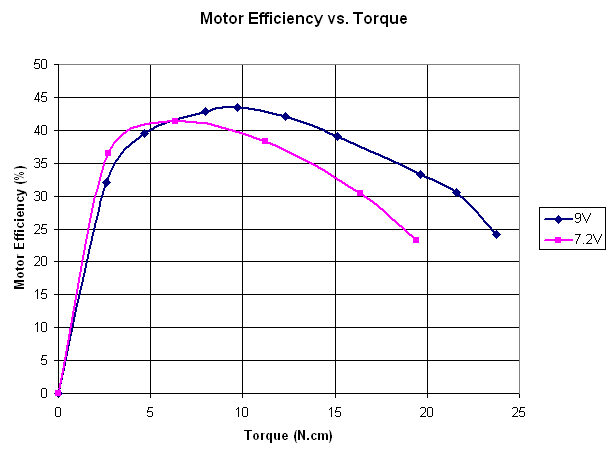
|  |  |
| --- | --- |
|  | http://www.philohome.com/nxtmotor/c1.gif Motor no cargado, potencia de 9V NXT. Lo más notable aquí es la relación lineal entre el nivel de potencia y la velocidad del motor. Mucho más práctico que lo que obtuvimos con RCX, donde el motor descargado corría casi a toda velocidad en el nivel de potencia 2. Como no hay carga que superar, el Control de potencia no tendría ningún efecto aquí. |
| http://www.philohome.com/nxtmotor/c2.gif El motor no está cargado, pero esta vez el NXT está alimentado a 7.2V (lo que obtienes con las baterías de NiMH). Por supuesto, la velocidad de rotación es proporcionalmente más baja. |
| http://www.philohome.com/nxtmotor/c3.gif Comportamiento del motor con  una carga de 11.5 N.cm aplicada, sin control de potencia, potencia de 9V NXT. Por debajo del 40% del motor está estancado (región horizontal de la curva). Una vez que se aplica suficiente potencia, la velocidad aumenta proporcionalmente. |
| http://www.philohome.com/nxtmotor/c4.gif Motor cargado con una potencia 11.5 N.cm, 9V NXT. Esta curva muestra la eficiencia del control de potencia: hasta el 70% de la velocidad es la misma que la de un motor sin carga. Después de eso, la curva es plana, el motor está funcionando a plena potencia. Tenga en cuenta que el movimiento como 10% fue muy irregular, incluso si el promedio de rpm está bien. |
| * Motor cargado con una potencia 11.5 N.cm, 7.2V NXT. Hasta el 50% de la velocidad es la misma que la de un motor sin carga de 9V (¡y en realidad más rápido que un motor con alimentación de 7.2V sin control de potencia!). La meseta aparece en un ciclo de trabajo inferior, ya que hay menos energía disponible. * http://www.philohome.com/nxtmotor/nxt_mot_chart.gif |

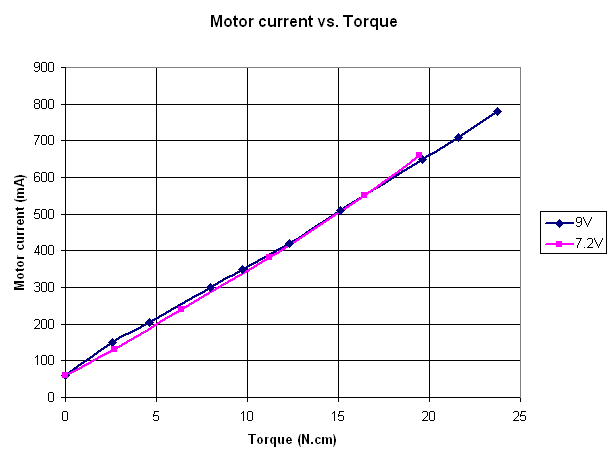
Los siguientes cuadros muestran las características del motor NXT frente a la carga aplicada. Para las curvas de color azul oscuro, el NXT fue alimentado a 9V (voltaje de las baterías alcalinas), el magenta se obtuvieron a 7.2V (voltaje de las baterías de NiMH). El nivel de potencia es 100% para todos los gráficos.



Esta curva muestra que la potencia mecánica máxima se obtiene con una carga de par de aproximadamente 15 N.cm. Si se compara con las [curvas obtenidas para el RCX con motor 71427](http://www.philohome.com/motors/mechpwr-rcx-71427.gif) , verá que la potencia mecánica disponible es mucho más alta, ¡casi 4 veces! Incluso alimentado con baterías NiMH de 7.2V, el NXT puede entregar más potencia que una salida RCX con 2 motores paralelos y suministro de 9V. Esto viene con un precio, por supuesto, la corriente drenada en ese nivel de potencia es mucho mayor, es mejor tener buenas baterías ...







La corriente frente al par muestra un aumento lineal con la carga. Debido a las limitaciones de potencia en el controlador NXT y la corriente de disparo del termistor en el motor NXT, le sugiero que no exceda un par de 15 N.cm durante períodos de tiempo prolongados. Las cargas más altas (por lo tanto, los drenajes de corriente) son posibles por períodos cortos, pero las protecciones pronto reducirán la corriente y la potencia disponible.

Sensor

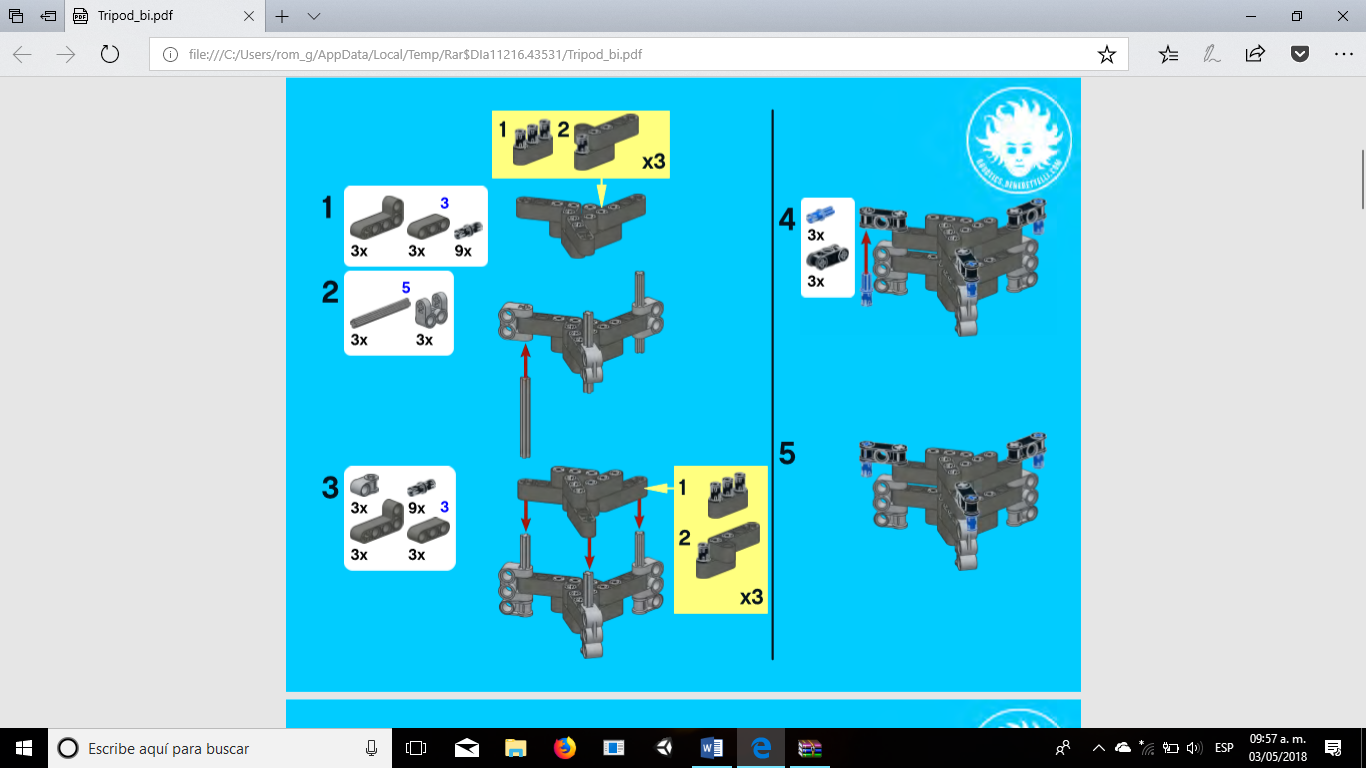
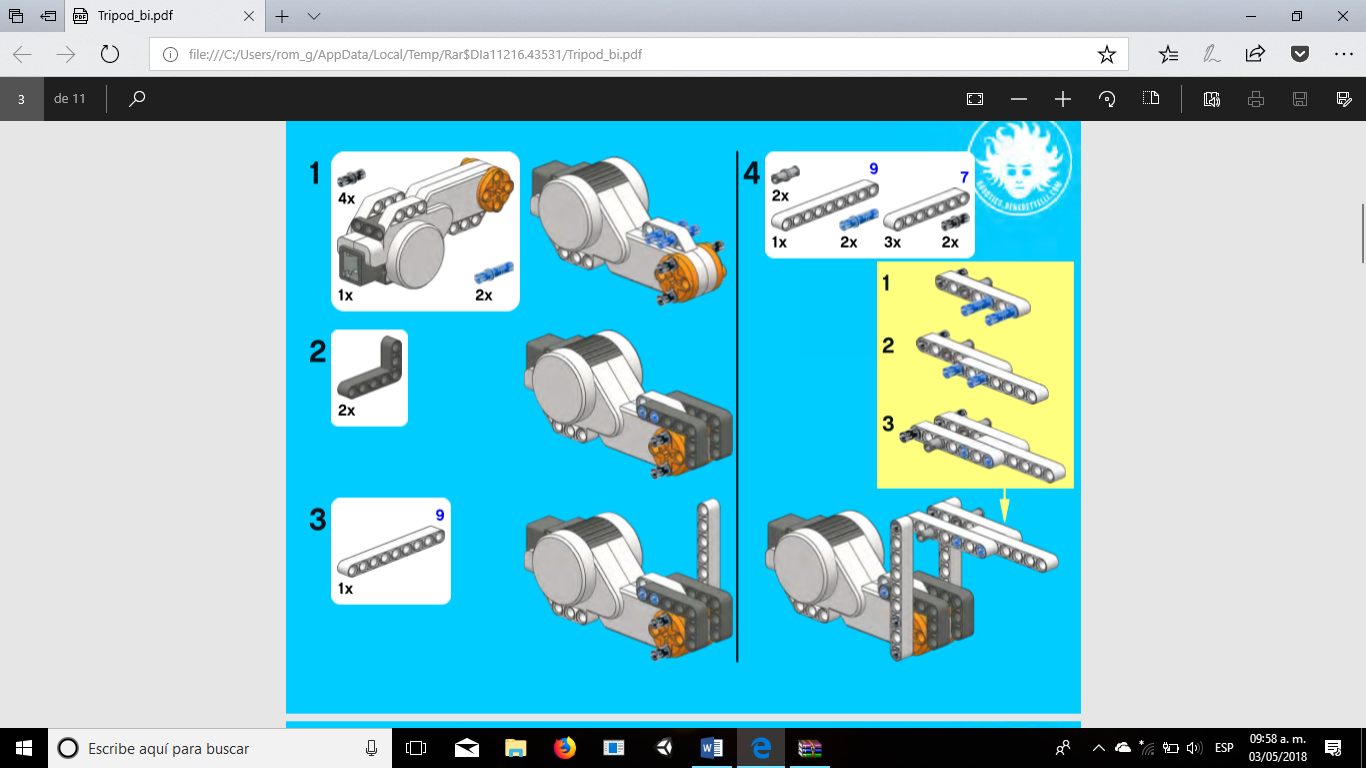
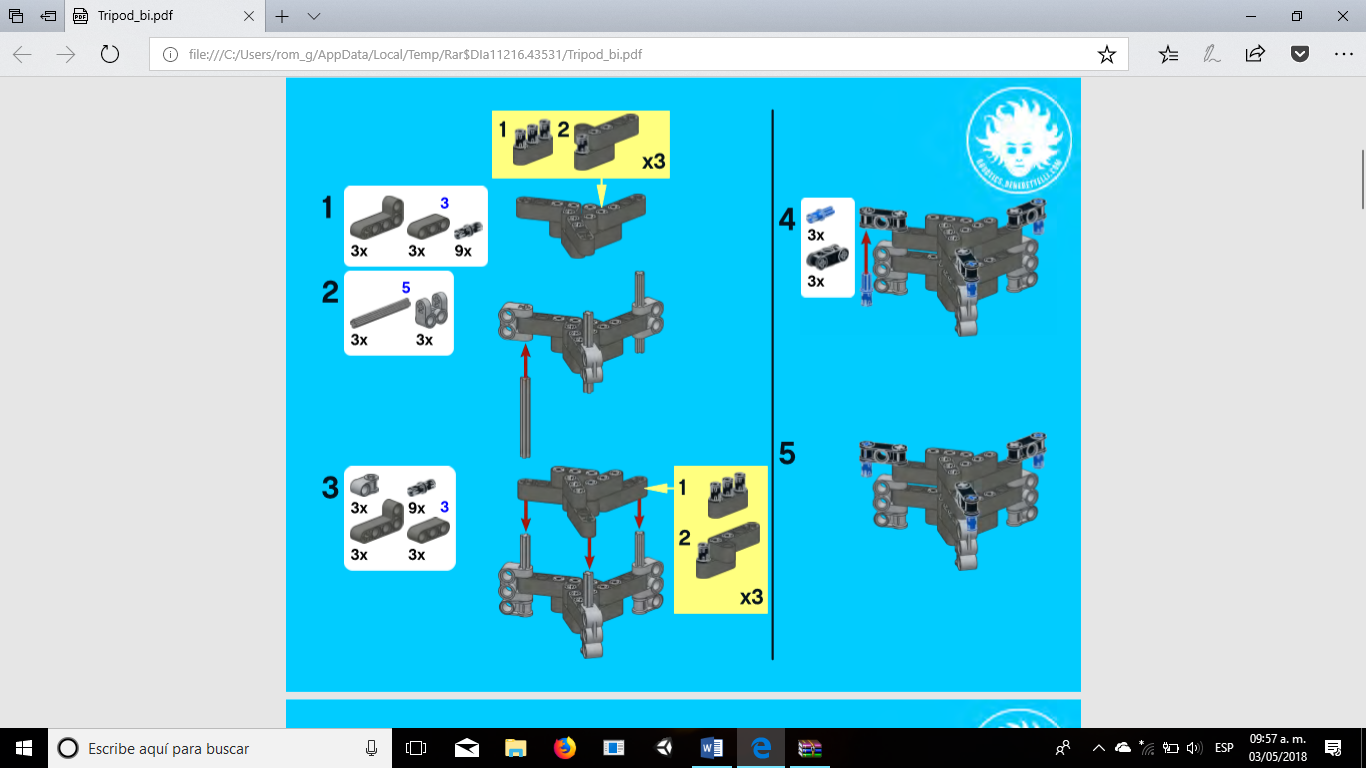
The Ultrasonic Sensor is one of the two sensors that give your robot “vision” [The Light Sensor is the other]. The Ultrasonic Sensor enables your robot to see and detect objects. You can also use it to make your robot avoid obstacles, sense and measure distance, and detect movement.

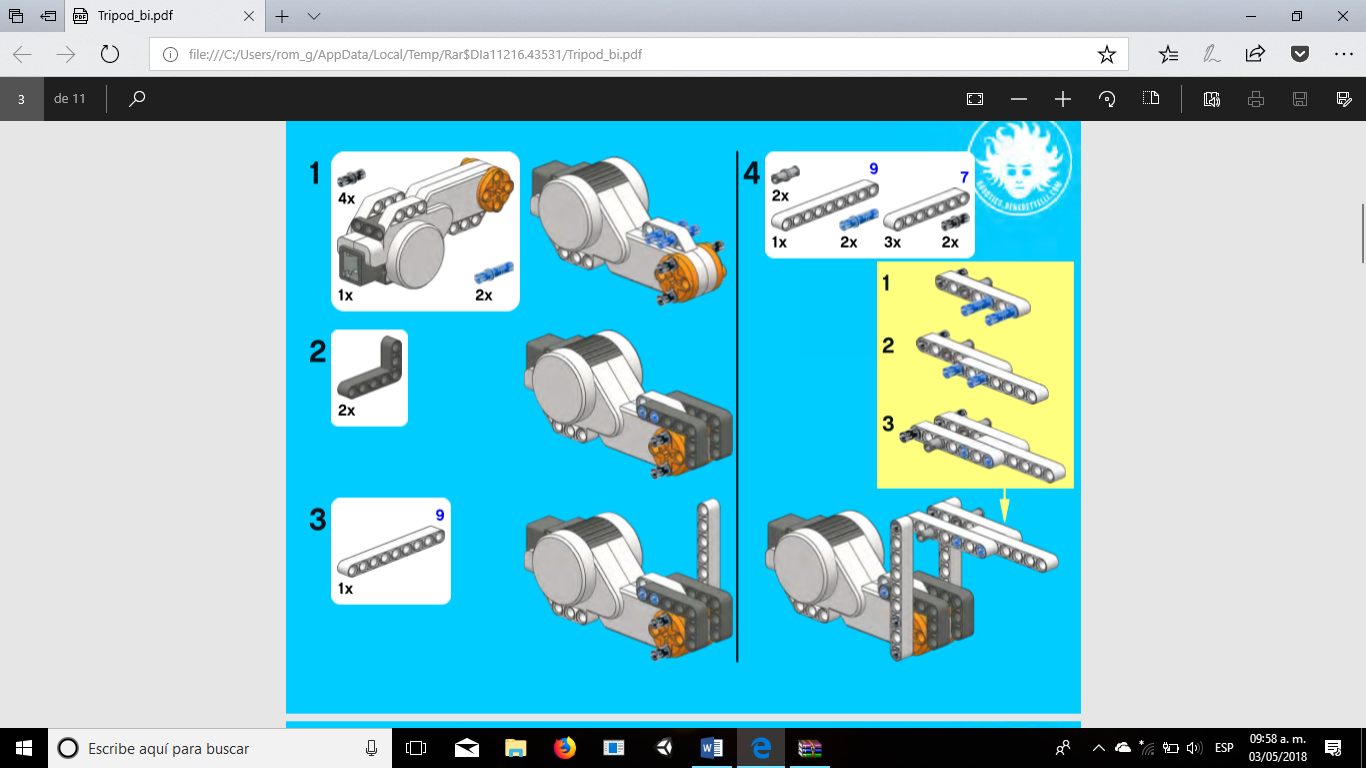
The Ultrasonic Sensor measures distance in centimeters and in inches. It is able to measure distances from 0 to 255 centimeters with a precision of +/- 3 cm.

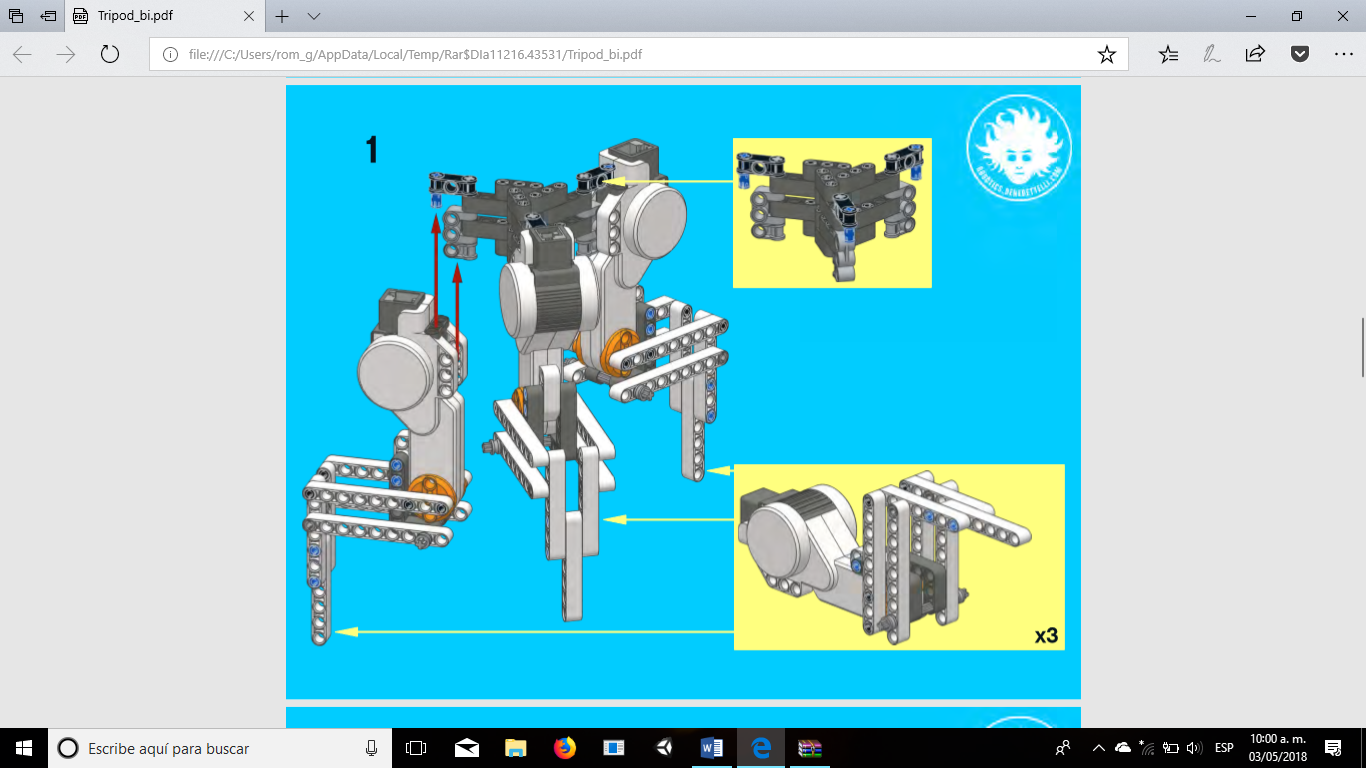
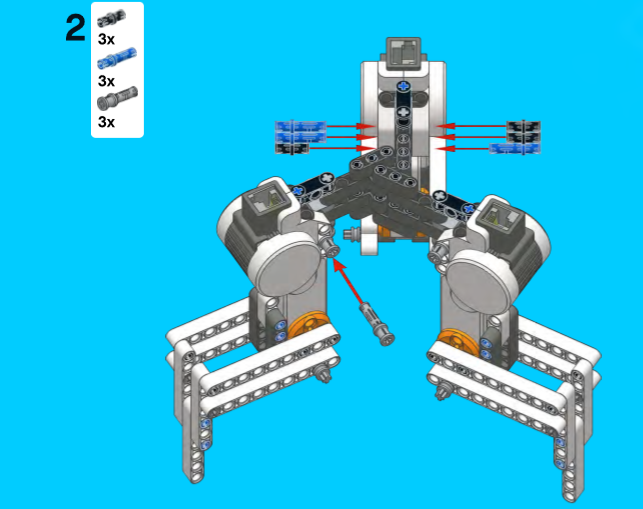
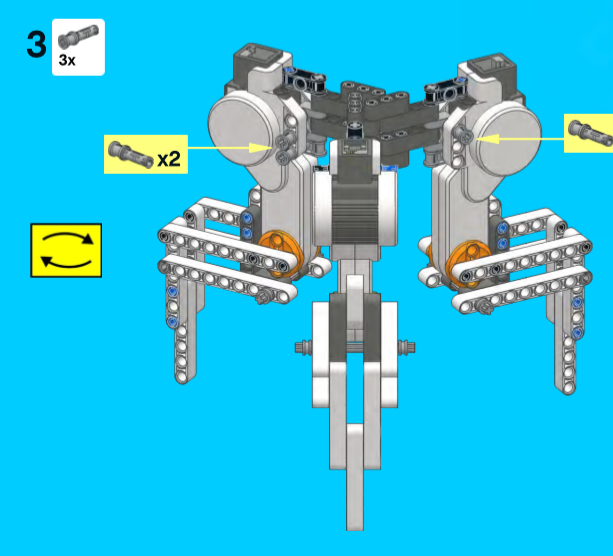
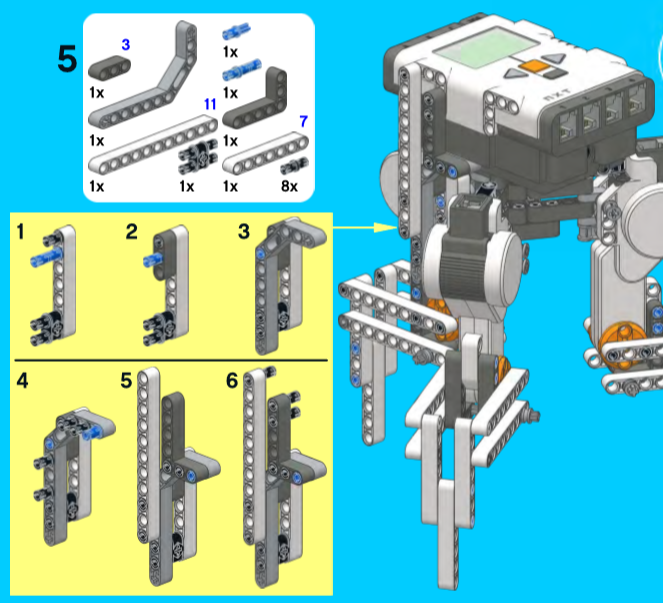
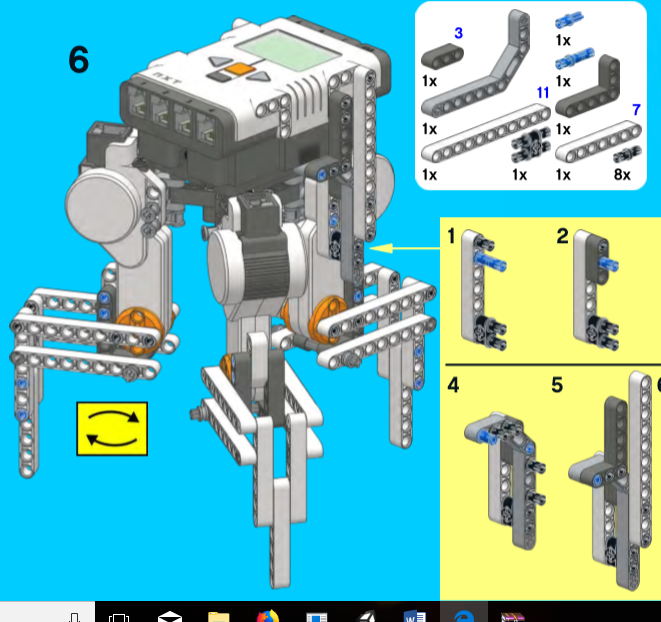
The Ultrasonic Sensor uses the same scientific principle as bats: it measures distance by calculating the time it takes for a sound wave to hit an object and return – just like an echo.

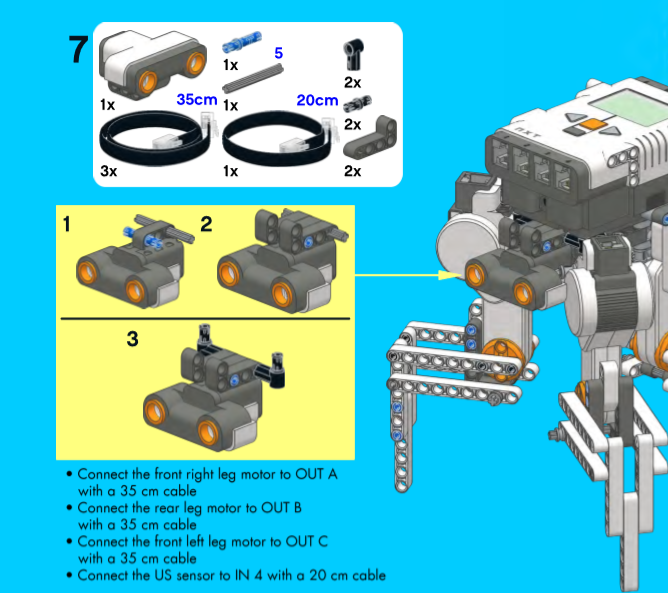
Large sized objects with hard surfaces return the best readings. Objects made of soft fabric or that are curved [like a ball] or are very thin or small can be difficult for the sensor to detect.







**Tripod Código**

#define STEP\_PAUSE 80

// Definiendo las extremidades del Tripod

#define LEFT\_LEG OUT\_C

#define REAR\_LEG OUT\_B

#define RIGHT\_LEG OUT\_A

#define EYE IN\_4

// Definiendo Direcciones

#define NEAR 25

#define NONE -1

#define LEFT 1

#define RIGHT 2

#define FRONT 3

#define BACK 4

#define CW 1

#define CCW 0

#define KP 30

#define KD 10

#define KI 0

#define DAMP 60

#define SCALE 5

#define MID 60

int ref[3];

//Ajuste de Extremidad (Piernas)

void Set(byte rear, byte left, byte right)

{

ref[REAR\_LEG] = rear;

ref[LEFT\_LEG] = left;

ref[RIGHT\_LEG] = right;

}

void Pulse(byte motor, int times, bool long\_pulse)

{

repeat (times)

{

ref[motor] = MID+10;

Wait(80);

//Wait(70+long\_pulse\*30);

ref[motor] = MID-10;

Wait(80);

}

}

void PulseDouble(byte motor\_one, byte motor\_two, int times, bool long\_pulse)

{

repeat (times)

{

ref[motor\_one] = 80;

ref[motor\_two] = 80;

Wait(50+long\_pulse\*40);

ref[motor\_one] = 40;

ref[motor\_two] = 40;

}

}

// Funcion de avanzar

void Walk (int dir, int times)

{

repeat(times)

{

//Si se dirige al frente

if (dir == FRONT)

{

Set(MID-20,MID,MID);

Wait(300);

Set(MID+10,MID,MID);

Wait(200);

}

//Si se dirige hacia atras

else if (dir == BACK)

{

Set(MID-10,MID+20,MID+20);

Wait(150);

Set(MID-10,MID-10,MID-10);

Wait(300);

}

//Si se dirige hacia la derecha

else if (dir == RIGHT)

{

Set(MID-10,MID-20,MID-10);

Wait(80);

Set(MID+10,MID-20,MID+10);

Wait(150);

Set(MID-10,MID-20,MID-10);

Wait(300);

}

// O si se dirige a la izquierda

else if (dir == LEFT)

{

Set(MID-10,MID-10,MID-20);

Wait(80);

Set(MID+10,MID+10,MID-20);

Wait(150);

Set(MID-10,MID-10,MID-20);

Wait(300);

}

// if (times>1) Wait(STEP\_PAUSE);

}

}

// Funciones para regresar en caso de obstaculo

void Turn(int wise, int times)

{

repeat (times)

{

ref[REAR\_LEG] = MID-10;

ref[wise==CCW? RIGHT\_LEG:LEFT\_LEG] = MID+10;

Wait(100);

ref[wise==CCW? LEFT\_LEG:RIGHT\_LEG] = MID+10;

ref[wise==CCW? RIGHT\_LEG:LEFT\_LEG] = MID-10;

Wait(100);

ref[wise==CCW? LEFT\_LEG:RIGHT\_LEG] = MID-10;

ref[REAR\_LEG] = MID+10;

Wait(100);

// Pulse( wise==CCW? REAR\_LEG:LEFT\_LEG, 1, false);

// Pulse(RIGHT\_LEG,1,false);

// Pulse( wise==CCW? LEFT\_LEG:REAR\_LEG,1,false);

}

}

void Demo()

{

Walk(FRONT,10);

Turn(CCW,10);

Walk(FRONT,10);

Turn(CW,10);

Walk(FRONT,10);

Walk(LEFT,10);

Walk(RIGHT,10);

}

// Control del motor A

task motorAcontroller()

{

int err, P, I, D, PID, Apos, Aold;

ResetAllTachoCounts(OUT\_A);

ResetRotationCount(OUT\_A);

while(true)

{

Apos = MotorRotationCount(OUT\_A);

err = Apos - ref[OUT\_A];

P = KP\*err;

I = err + KI\*I\*(100-DAMP)/100;

D = KD\*(err-Aold);

PID = (P + I + D)/SCALE;

PID = abs(PID)>100? sign(PID)\*100 : PID;

OnRev(OUT\_A,PID);

Aold = err;

}

}

// Control del motor B

task motorBcontroller()

{

int err, P, I, D, PID, Bpos, Bold;

ResetAllTachoCounts(OUT\_B);

ResetRotationCount(OUT\_B);

while(true)

{

Bpos = MotorRotationCount(OUT\_B);

err = Bpos - ref[OUT\_B];

P = KP\*err;

I = err + KI\*I\*(100-DAMP)/100;

D = KD\*(err-Bold);

PID = (P + I + D)/SCALE;

PID = abs(PID)>100? sign(PID)\*100 : PID;

OnRev(OUT\_B,PID);

Bold = err;

}

}

// Control del motor C

task motorCcontroller()

{

int err, P, I, D, PID, Cpos, Cold;

ResetAllTachoCounts(OUT\_C);

ResetRotationCount(OUT\_C);

while(true)

{

Cpos = MotorRotationCount(OUT\_C);

err = Cpos - ref[OUT\_C];

P = KP\*err;

I = err + KI\*I\*(100-DAMP)/100;

D = KD\*(err-Cold);

PID = (P + I + D)/SCALE;

PID = abs(PID)>100? sign(PID)\*100 : PID;

OnRev(OUT\_C,PID);

Cold = err;

}

}

void triped\_init()

{

//SetSensorLowspeed(LEFT\_EYE);

//SetSensorLowspeed(RIGHT\_EYE);

SetSensorLowspeed(EYE);

OnRev(OUT\_ABC,20);

Wait(80);

Float(OUT\_ABC);

Wait(500);

start motorAcontroller;

start motorBcontroller;

start motorCcontroller;

//Set(MID+40,MID+40,MID+40);

//Wait(1000);

Set(MID,MID,MID);

Wait(1000);

/\*

repeat(3)

{

Set(MID,MID+20,MID+20);

Wait(200);

Set(MID+20,MID+20,MID);

Wait(200);

Set(MID+20,MID,MID+20);

Wait(600);

}

Set(MID,MID,MID);

\*/

}

//Brincos

void Jump( int times )

{

repeat (times)

{

Set(90,90,90);

Wait(1000);

for (int i=90; i >=30; i--)

{

Set(i,i,i);

Wait(1);

}

Wait(1000);

}

}

task main()

{

int count;

triped\_init();

while (true)

{

if (ButtonPressed(BTNCENTER,true))

{

Set(MID+50,MID+50,MID+50);

while(ButtonPressed(BTNCENTER,true));

until(ButtonPressed(BTNCENTER,true));

}

if (SensorUS(EYE)>NEAR)

{

Walk(FRONT,1);

}

else

{

Walk(BACK,3);

count = 0;

while(SensorUS(EYE)<=NEAR && count<20)

{

Turn(CCW,3);

count++;

}

if (SensorUS(EYE)<=NEAR)

{

count = 0;

while(SensorUS(EYE)<NEAR && count<20)

{

Turn(CW,3);

count++;

}

Walk(LEFT,4);

}

else

{

Walk(RIGHT,4);

}

//Turn(CCW,5);

}

}

}