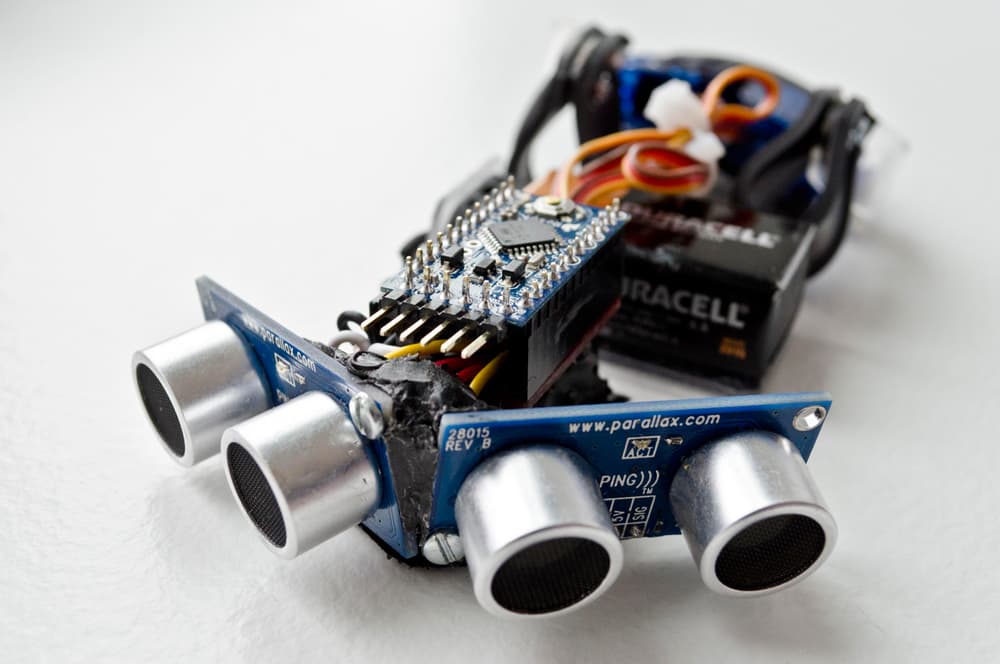
Programación de Microcontroladores con Tecnologías Libres



|  |  |
| --- | --- |
| Fecha | 14 de Julio de 2016 |
| Proyecto Final | Tacit |
| Integrantes | Presa, Federico  Schafir, Esteban |

Índice

[Objetivo 3](#_Toc456249174)

[Introducción 3](#_Toc456249175)

[Componentes 3](#_Toc456249176)

[Materiales secundarios 3](#_Toc456249177)

[Desarrollo 4](#_Toc456249178)

[Armado del guante 12](#_Toc456249179)

Objetivo

Basado en el proyecto “Tacit”[[1]](#footnote-1), el objetivo del presente trabajo es crear, mediante hardware Arduino, un accesorio para las manos que indica a través de vibraciones cuando un objeto está próximo de alcance.

Introducción

“Tacit” es un sistema portátil que traduce la distancia de cualquier cosa que apunte con la mano, hacia una presión en la muñeca. Cuanto más cerca del objeto, mayor será la presión. Si mueve su mano alrededor, el dispositivo transmite una imagen táctil de su entorno. Este proyecto fue diseñado para ayudar a las personas con problemas de visión a desenvolverse en el ambiente, pero es también una diversión y mejora sensorial eficaz para las personas videntes - especialmente en la oscuridad.

Componentes

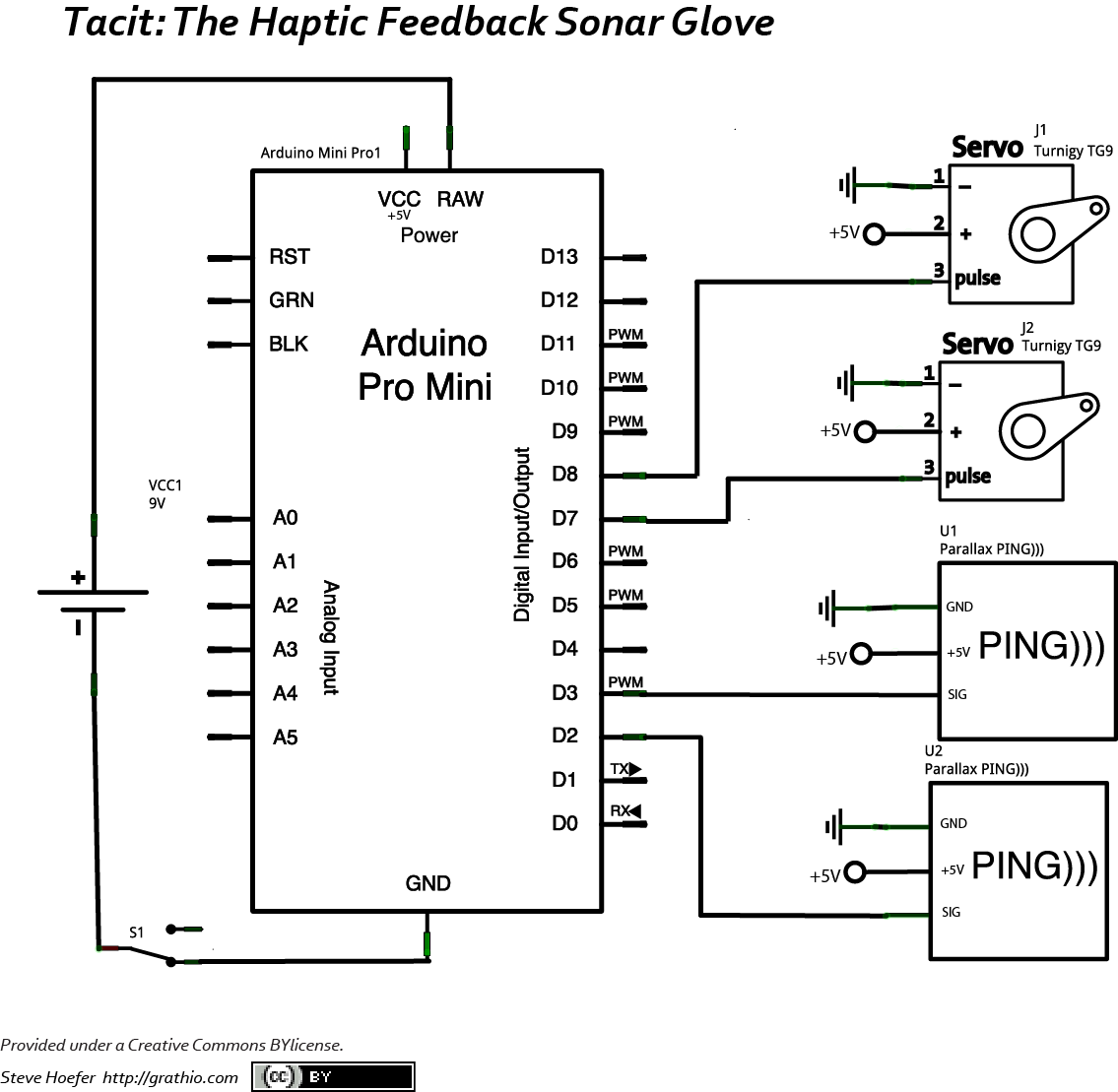
* x1 Arduino UNO
* x1 Protoboard
* x2 Sensor ultrasónicos HC-SR04[[2]](#footnote-2)
* x2 Mini Servo Tower Pro SG90 9g[[3]](#footnote-3)
* x1 Cable Adaptador para batería (9v)
* x1 Caja de cartón
* x1 Cajita de fósforos
* x1 Bateria (9V)

Materiales secundarios

* Adhesivos
* Hilo de coser
* Aguja
* Goma eva
* Suprabond
* Abrojo

Desarrollo

Se comenzó el desarrollo del proyecto conectando los componentes en el ARDUINO UNO tal como lo indica el siguiente diagrama:



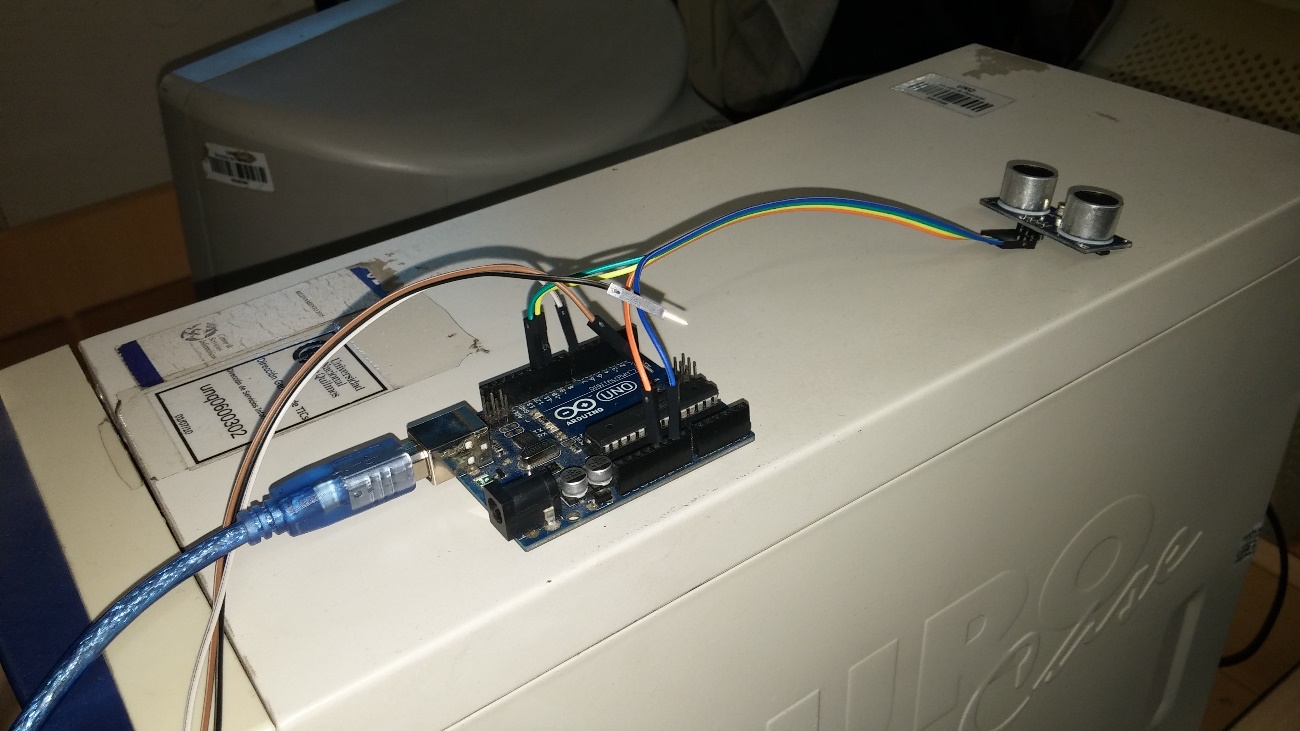
Debido a que el diagrama está basado en un Arduino Pro Mini, se tuvo que ajustar el diagrama para un Arduino UNO. Además de diferir en el hardware de Arduino, ciertos componentes utilizados en el proyecto original diferían de los utilizados por nosotros presentando otra dificultad al momento de ensamblar. Particularmente ocurrió con los sensores de ultrasonido. A continuación se observa el sensor utilizado en el proyecto original vs el utilizado por nosotros:



Original: Sensor ultrasónico de 3 pines



Utilizado: Sensor ultrasónico de 4 pines



Sensor ultrasónico conectado

Una vez finalizada la conexión de los componentes, se descargó desde la página oficial el código fuente y se cargó en el Arduino UNO. El siguiente código pertenece al repositorio oficial:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | #include <Servo.h> | | const int MaxSensors = 2; // The number of sensor/servo pairs. | | const int ServoPins[MaxSensors] = {7, 8}; // The pins they're on | | const int RangingPins[MaxSensors] = {3, 2}; // The pins they're on | | const int ReadingsPerSensor = 5; // The number of historic readings to consider when determining position. | | const int TimePerDegree = 9; // ms per degree rotation on the servo to prevent servo motor electrical noise from interfering with the ultrasonic sensor readings | | const int MinimumTurnDistance = 3; // Minimum number of degrees that the servo will turn. Keeps the servos from being too twitchy. | | // Variables | | Servo ServoList[MaxSensors]; // Array of servo objects for manipulating easily. | | int sensorReadings[MaxSensors][ReadingsPerSensor]; // Hold past readings for each sensor. | | int calculatedSenorReadings[MaxSensors]; // The calculated distance for each sensor. | | int latestReading = 0; // Current position in the array for the most recent reading. | | int servoLocations[MaxSensors]; // The current position of each sensor. | | int SenorClose = 500; // Closest value we detect with the PING sensor. (Soundwave travel time in milliseconds.) | | int SensorFar = 14000; // Furthest distance we register on the PING sensor. (Soundwave travel time in milliseconds.) | | int ServoClose[MaxSensors] = {0, 160}; // Angle the servo turns to when something is closest. | | int ServoFar[MaxSensors] = {70,110}; // Angle the servo turns to when something is at its furthest. | |  | | void setup() { | |  | | // Initialize the servo location and move them through a full range of motion so we know they work. | | for (int i = 0; i < MaxSensors; i++){ | | ServoList[i].attach(ServoPins[i]); | | delay(10); | | ServoList[i].write(ServoClose[i]); | | delay(500); | | ServoList[i].write(ServoFar[i]); | | delay(500); | | ServoList[i].detach(); | | } | | delay(100); | |  | | } | |  | | void loop(){ | | int i, j, oldLocation; | | unsigned long delayTime; | |  | | // Loop through each range sensor | | for (i = 0; i < MaxSensors; i++){ | | // Get the current sensor's range. | | sensorReadings[i][latestReading] = getDistance(i); | | // Figure out an averaged/smoothed readings based on this and past data. | | calculatedSenorReadings[i] = calculateNewDistace(i); | |  | | // Set the servo to the correct angle. | | oldLocation = servoLocations[i]; | | servoLocations[i] = map(calculatedSenorReadings[i], 0, 100, ServoClose[i], ServoFar[i]); | |  | | if (latestReading >= ReadingsPerSensor-1){ // Don't do anything until we have enough data to trend. | | if (abs(servoLocations[i]-oldLocation) >= MinimumTurnDistance){ // Only try to turn it if we have somewhere to go. | | ServoList[i].attach(ServoPins[i]); | | delay(10); | | ServoList[i].write(servoLocations[i]); | | delayTime = (TimePerDegree \* (abs(servoLocations[i]-oldLocation))+20); // Set a delay for the next reading so motor noise doesn't interfere with senor readings. | | if (abs(delayTime)>500){ // If it can't do it in this amount of time // It's based on how far it has to turn to keep the delay to a minimum, response time at a maximum. | | delayTime=500; // we'll get it next time. Keep it responsive. | | } | | delay(delayTime); | | ServoList[i].detach(); | | } else { // Otherwise if the reading hasn't changed enough write the old value to | | ServoList[i].attach(ServoPins[i]); // the servo so that it will hold in place if it's applying pressure. | | delay(10); | | ServoList[i].write(oldLocation); | | delay(50); | | ServoList[i].detach(); | | servoLocations[i]=oldLocation; | | } | | } | | delay(20); // Added to fix left sensor misbehavior reported by Rob. | | } | |  | | latestReading++; // Increment the reading counter so we know where we're at. | | if (latestReading >= ReadingsPerSensor){ // Make sure we don't record more readings than we have space to hold. | | latestReading = ReadingsPerSensor-1; | | // Pop the oldest reading off the list. | | for (i = 0; i < MaxSensors; i++){ | | for (j=0; j < ReadingsPerSensor-1; j++){ | | sensorReadings[i][j] = sensorReadings[i][j+1]; | | } | | } | | } | | } | |  | | // function: calculateNewDistace(sensorNumber: Which sensor's data to process): Calculated distance in 0-100 range. | | // Apply some averaging and smoothing to the recorded distance readings | | // to take care of noisy data. | | int calculateNewDistace(int sensorNumber){ | | int output = SensorFar; // Default value is the furthest distance. | |  | | float weightingFactor = 0.5; // How fast the reading's importance tapers off in time. (1= no taper, 0 = divide by zero error.) | | float flickerFactor = 30; // When the change is greater than this, ignore it unless its two in a row. (It's probably noise.) | |  | | if (latestReading >= ReadingsPerSensor-1) { // Only do this if we have a full set of readings to sample. | | int total = 0; // Average them with a weighting. | | float currentWeight = 1; // New readings count more than older readings. | | float percentagePossible = 0; | | boolean flickered = false; | | for (int i=ReadingsPerSensor-1; i >=0 ;i--){ // Check for flicker (This reduces jitter with something right on the threshold.) | | flickered = false; | | if (i==ReadingsPerSensor-1){ | | if ((abs(sensorReadings[sensorNumber][i])-abs(sensorReadings[sensorNumber][i-1]) > flickerFactor) && | | (abs(sensorReadings[sensorNumber][i-1])-abs(sensorReadings[sensorNumber][i-2]) > flickerFactor)){ | | flickered = true; | | } | | } | | if (flickered==false){ | | total += (sensorReadings[sensorNumber][i] \* currentWeight); | | percentagePossible += currentWeight; | | currentWeight \*= weightingFactor; | | } | | } | | output = total / percentagePossible; | | } | | return output; | | } | | // function: getDistance | | // Take a sensor number (not pin number) and returns an int in the 0-100 range | | // 0 = closest, 100= furthest. (It's a percentage of the distance that the software | | // | | // Note: Function is designed to be generic so that it can be swapped out for | | // different kinds of ranging sensors. | | // This version of the function is made for Parallax PING))) sensors | | // For more info see http://arduino.cc/en/Tutorial/Ping | | // and http://www.parallax.com/tabid/768/ProductID/92/Default.aspx | | int getDistance(int sensorNumber){ | | long duration; // How long it takes a sonic pulse to reflect back. | | int out; // The value we send back from the function | |  | | // Initialize the sensor and tell it to send out a ping. | | pinMode(RangingPins[sensorNumber], OUTPUT); | | digitalWrite(RangingPins[sensorNumber], LOW); | | delayMicroseconds(2); | | digitalWrite(RangingPins[sensorNumber], HIGH); | | delayMicroseconds(5); | | digitalWrite(RangingPins[sensorNumber], LOW); | |  | | // Read the time in milliseconds until the value comes back. | | pinMode(RangingPins[sensorNumber], INPUT); | | duration = pulseIn(RangingPins[sensorNumber], HIGH); | |  | | // Trim the data into minimums and maximums and map it to the 0-100 output range. | | duration = constrain(duration, SenorClose, SensorFar); | | out = map(duration, SenorClose, SensorFar, 0, 100); | | return out; | | } | |

Como mencionamos previamente acerca del hardware utilizado y dando el ejemplo de los sensores ultrasónicos, el código fuente debía ser ajustado para poder hacer funcionar todas las partes correctamente. Para ello se tuvo que descargar un módulo llamado “Ultrasonido”[[4]](#footnote-4) para incorporarlo en el código. El siguiente código es el resultado de realizar ajustes al código inicial y es el utilizado actualmente en el proyecto realizado:

|  |
| --- |
| #include <Servo.h>  #include<Ultrasonido.h>  const int MaxSensors = 2; // The number of sensor/servo pairs.  const int ServoPins[MaxSensors] = {7, 8}; // The pins they're on  Ultrasonido RangingPins[MaxSensors] = {Ultrasonido(10,11,100), Ultrasonido(12,13,100)}; // The pins they're on  const int ReadingsPerSensor = 5; // The number of historic readings to consider when determining position.  const int TimePerDegree = 9; // ms per degree rotation on the servo to prevent servo motor electrical noise from interfering with the ultrasonic sensor readings  const int MinimumTurnDistance = 3; // Minimum number of degrees that the servo will turn. Keeps the servos from being too twitchy.  // Variables  Servo ServoList[MaxSensors]; // Array of servo objects for manipulating easily.  int sensorReadings[MaxSensors][ReadingsPerSensor]; // Hold past readings for each sensor.  int calculatedSenorReadings[MaxSensors]; // The calculated distance for each sensor.  int latestReading = 0; // Current position in the array for the most recent reading.  int servoLocations[MaxSensors]; // The current position of each sensor.  int SenorClose = 500; // Closest value we detect with the PING sensor. (Soundwave travel time in milliseconds.)  int SensorFar = 14000; // Furthest distance we register on the PING sensor. (Soundwave travel time in milliseconds.)  int ServoClose[MaxSensors] = {0, 160}; // Angle the servo turns to when something is closest.  int ServoFar[MaxSensors] = {70,110}; // Angle the servo turns to when something is at its furthest.  void setup() {  Serial.begin(9600);  for (int i = 0; i < MaxSensors; i++){  ServoList[i].attach(ServoPins[i]);  delay(10);  ServoList[i].write(ServoClose[i]);  delay(500);  ServoList[i].write(ServoFar[i]);  delay(500);  ServoList[i].detach();  }  delay(100);  }  void loop(){  int i, j, oldLocation;  unsigned long delayTime;  // Loop through each range sensor  for (i = 0; i < MaxSensors; i++){  // Get the current sensor's range.  sensorReadings[i][latestReading] = getDistance(i);  // Figure out an averaged/smoothed readings based on this and past data.  calculatedSenorReadings[i] = calculateNewDistace(i);  // Set the servo to the correct angle.  oldLocation = servoLocations[i];  servoLocations[i] = map(calculatedSenorReadings[i], 0, 100, ServoClose[i], ServoFar[i]);  if (latestReading >= ReadingsPerSensor-1){ // Don't do anything until we have enough data to trend.  if (abs(servoLocations[i]-oldLocation) >= MinimumTurnDistance){ // Only try to turn it if we have somewhere to go.  ServoList[i].attach(ServoPins[i]);  delay(10);  ServoList[i].write(servoLocations[i]);  delayTime = (TimePerDegree \* (abs(servoLocations[i]-oldLocation))+20); // Set a delay for the next reading so motor noise doesn't interfere with senor readings.  if (abs(delayTime)>500){ // If it can't do it in this amount of time // It's based on how far it has to turn to keep the delay to a minimum, response time at a maximum.  delayTime=500; // we'll get it next time. Keep it responsive.  }  delay(delayTime);  ServoList[i].detach();  } else { // Otherwise if the reading hasn't changed enough write the old value to  ServoList[i].attach(ServoPins[i]); // the servo so that it will hold in place if it's applying pressure.  delay(10);  ServoList[i].write(oldLocation);  delay(50);  ServoList[i].detach();  servoLocations[i]=oldLocation;  }  }  delay(20); // Added to fix left sensor misbehavior reported by Rob.  }  latestReading++; // Increment the reading counter so we know where we're at.  if (latestReading >= ReadingsPerSensor){ // Make sure we don't record more readings than we have space to hold.  latestReading = ReadingsPerSensor-1;  // Pop the oldest reading off the list.  for (i = 0; i < MaxSensors; i++){  for (j=0; j < ReadingsPerSensor-1; j++){  sensorReadings[i][j] = sensorReadings[i][j+1];  }  }  }  }  int calculateNewDistace(int sensorNumber){  int output = SensorFar; // Default value is the furthest distance.  float weightingFactor = 0.5; // How fast the reading's importance tapers off in time. (1= no taper, 0 = divide by zero error.)  float flickerFactor = 30; // When the change is greater than this, ignore it unless its two in a row. (It's probably noise.)  if (latestReading >= ReadingsPerSensor-1) { // Only do this if we have a full set of readings to sample.  int total = 0; // Average them with a weighting.  float currentWeight = 1; // New readings count more than older readings.  float percentagePossible = 0;  boolean flickered = false;  for (int i=ReadingsPerSensor-1; i >=0 ;i--){ // Check for flicker (This reduces jitter with something right on the threshold.)  flickered = false;  if (i==ReadingsPerSensor-1){  if ((abs(sensorReadings[sensorNumber][i])-abs(sensorReadings[sensorNumber][i-1]) > flickerFactor) &&  (abs(sensorReadings[sensorNumber][i-1])-abs(sensorReadings[sensorNumber][i-2]) > flickerFactor)){  flickered = true;  }  }  if (flickered==false){  total += (sensorReadings[sensorNumber][i] \* currentWeight);  percentagePossible += currentWeight;  currentWeight \*= weightingFactor;  }  }  output = total / percentagePossible;  }  return output;  }  int getDistance(int sensorNumber){  int distancia=RangingPins[sensorNumber].Distancia();  Serial.println(sensorNumber\*1000+distancia);  return RangingPins[sensorNumber].Distancia();  } |

Ciertas funciones debieron ser modificadas para poder hacer funcionar correctamente el programa, como por ejemplo “getDistance” y también la declaración de los pines en “RangingPins”.

Una vez modificado el programa y testeado, solo restaba el maquetado. Entre idas y vueltas acerca del material que se iba a utilizar, se consideró que el neoprenne era un material muy costoso para el proyecto, teniendo en cuenta además que una vez conseguido habría que realizar ciertos trabajos de “manualidades” que podrían ser riesgosos y requerir de otro guante del mismo material en caso de fallar. Es por eso que se eligió utilizar en su reemplazo, un guante de goma.



Guante de goma utilizado

Armado del guante

1. Se trabajó con la caja en donde se va a incluir el protoboard y el Arduino uno. Esto resultó sencillo ya que lo que se hizo para que quedara firme fue utilizar adhesivos.



1. En la parte delantera se hizo un agujero horizontal finito para poder pasar todos los cables de los servomotores y los sensores ultrasónicos.
2. En la parte trasera de la caja se tuvo que hacer un agujero circular para darle espacio para la ficha de alimentación del proyecto.
3. Se continuó con la siguiente parte: la idea de los servomotores es para que se haga presión en la mano. Para ello necesitamos de un pedacito de acetato (plástico) para adaptarlo a las patas de los servos. A estos se le dio forma de L.



1. El siguiente paso fue armar una base más blanda para colocar arriba la caja con todos sus componentes. Lo que se decidió fue cortar dos rectángulos de goma eva y pegarlos con suprabond.



Goma eva



Goma eva con pegamento

1. Para unir el guante con la caja lo que se hizo fue coser la goma eva al guante de la siguiente manera (dos puntadas en los vértices y una en el medio)



Guante cosido

1. Luego se necesitaba de alguna forma dejar todo más organizado la parte de la caja para que no se vean los cables, etc. Lo que se hizo fue cortar una plancha rectangular que unimos a la base de la caja para poder de alguna manera envolverla de manera que tape la placa, el protoboard y los cables.



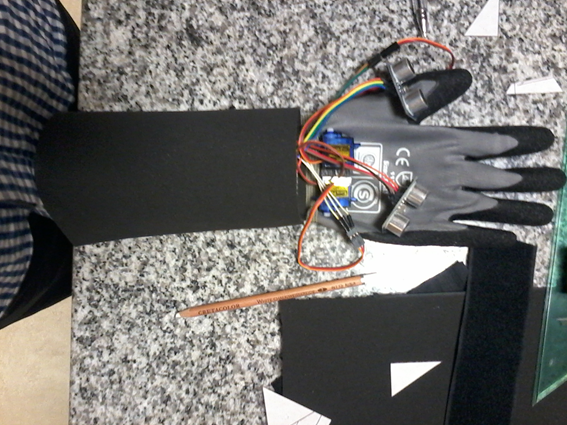
Guante y protoboard

1. Luego se incorporó una tira de abrojo recién cortada para poder sujetarlo del otro lado y pegamos otra tira sobre el lado derecho de la caja para pegarlo cuando se cierre.



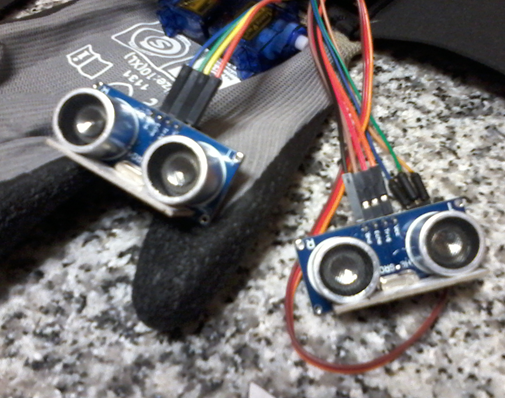
Abrojo pegado en la goma eva

1. Así quedaría:



Resultado de incorporar el abrojo

1. Ahora, se necesitaba que los sensores ultrasónicos queden sujetos al guante. Lo que se hizo fue hacerle una base de cartón a cada uno y pegarlo debajo del servo. Luego esa base se pegó al guante cerca de los nudillos así quedan apuntando hacia delante.



Sensores de ultrasonido antes de ser pegados

1. Al tener una caja pequeña como base del Arduino. Se tuvo que optar por poner la batería para alimentar el Arduino al costado. Esto se realizó con una cajita de fósforos pequeña. Se la cubrió con goma eva y se la pego al costado.



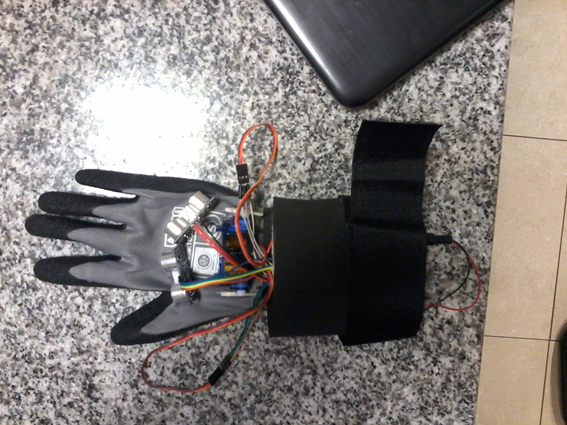
Sensores de ultrasonido incorporados al guante mediante la caja de fosforos.

1. Para finalizar, se necesitaba que quede el proyecto compacto al brazo de una persona. Para ello se deicidio cortar una tira de abrojo para que haga el agarre en el abrazo.



Abrojo incorporado

Resultado Final:



Resultado final del dispositivo

1. By Steve Hoefer at Grathio Labs ([http://grathio.com](http://grathio.com/)). URL: <https://gist.github.com/Grathio/1175994> [↑](#footnote-ref-1)
2. <http://articulo.mercadolibre.com.ar/MLA-620084391-sensor-ultrasonico-hc-sr04-para-arduino-pic-robotica-_JM> [↑](#footnote-ref-2)
3. http://articulo.mercadolibre.com.ar/MLA-619943977-mini-servo-tower-pro-sg90-9g-robotica-arduino-servomotor-\_JM [↑](#footnote-ref-3)
4. https://docs.google.com/file/d/0B0hsUkhqWH97XzhhQnRIQVdVeHM/edit [↑](#footnote-ref-4)