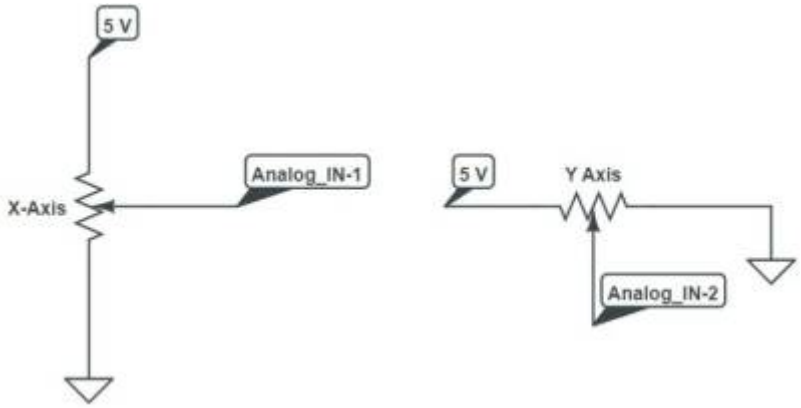
Joystick et Arduino

https://www.makerguides.com/tutorials-on-arduino-and-joystick-interface/

Comment fonctionne un joystick ?

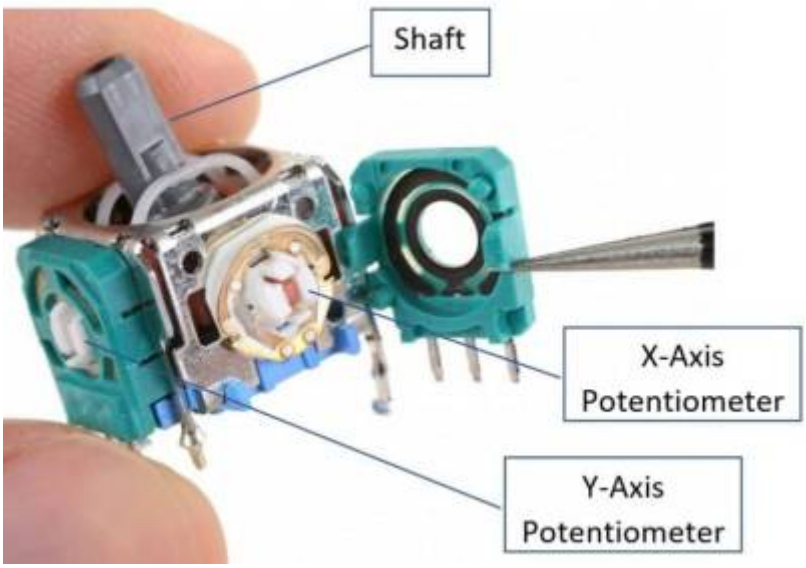
Le joystick est composé de deux potentiomètres. Un potentiomètre pour le mouvement de l'axe X et l'autre pour le mouvement de l'axe Y. Comment fonctionne un joystick



Les potentiomètres reposent toujours en position centrale. Si le potentiomètre est alimenté en 5 V, la valeur analogique mesurée au centre du potentiomètre sera de 2,5 V.

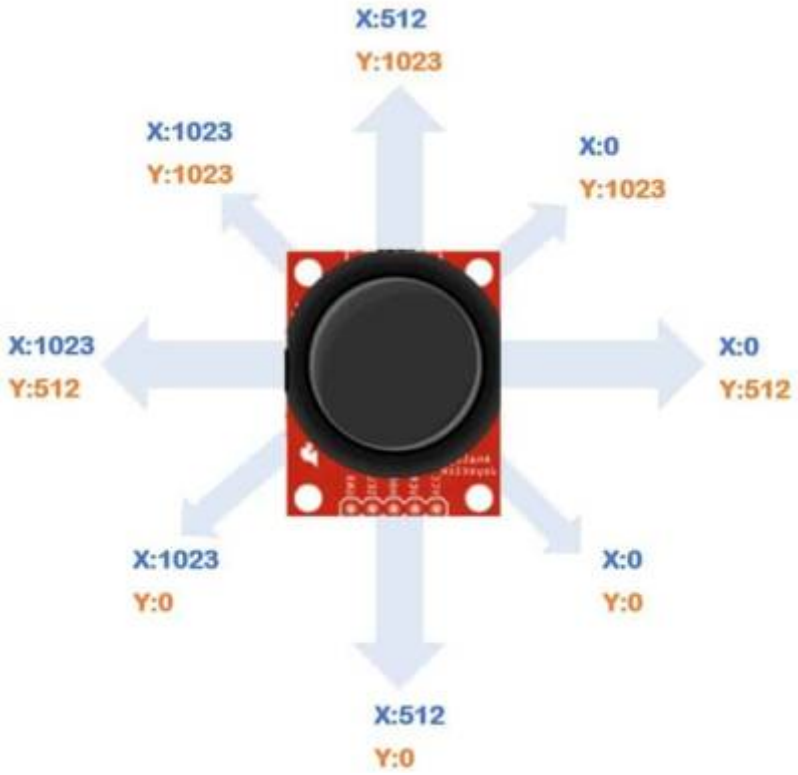
Dans l'image ci-dessous, vous pouvez voir les parties internes du joystick.

Parties internes du joystick Source :
https://www.ifixit.com/News/48944/heres-why-ps5-joysticks-drift-and-why-theyll-only-get-worse



Vous avez besoin de deux broches d'entrée analogiques sur l'Arduino UNO pour lire les données du joystick. J'ai créé des graphiques simples qui montrent les valeurs analogiques attendues que vous

lisez sur les deux potentiomètres.



Les deux potentiomètres tournent automatiquement au ralenti en position centrale. Cela signifie que la valeur lue sera la moitié de la résolution maximale de l'ADC.

Par exemple, si le bit ADC est de 10 bits, alors la valeur par défaut lue par les entrées analogiques sera 512.

La valeur ADC sera de 512 pour le potentiomètre en position de repos.

Dans l'image ci-dessus, vous pouvez voir le nombre d'ADC auquel vous pouvez vous attendre à différentes positions de l'arbre.

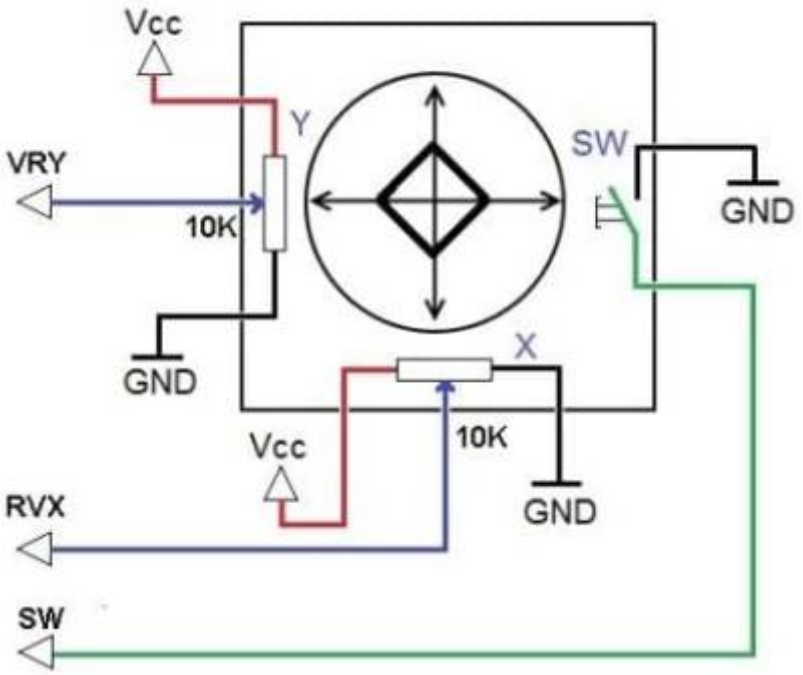
Lorsque vous avancez l'arbre, le potentiomètre de l'axe X reste inactif, mais le potentiomètre de l'axe Y avance complètement.

Par conséquent, les valeurs ADC pour les axes X et Y sont respectivement de 512 et 1024.

Les joysticks auront également un interrupteur momentané que vous pourrez utiliser pour des applications spécifiques.

Ainsi, un seul joystick suffit parfois pour effectuer diverses tâches.

Représentation électrique d'un joystick à deux axes Source : https://www.store-ino.com/?attachment_id=698



L'image ci-dessus montre une représentation électrique d'un joystick à deux axes. Vous pouvez également voir l’option de commutation.

Notez qu'il y a un potentiomètre de dix kOhms à l'intérieur du joystick. Tous les joysticks n'auront pas dix kOhms.

Cela n'a pas d'importance puisque ce que nous lisons comme tension analogique n'est que la différence relative de tension.

Voyons le brochage d'un joystick couramment disponible.

Sl. Non	Étiquette d'épingle	Fonction de broche
1	GND	Connexion à la terre
2	5V	Alimentation pour le joystick. Le 5 V fournit une tension de polarisation pour les potentiomètres
3	VRx	Sortie de tension analogique pour le potentiomètre de l'axe X
4	VRy	Sortie de tension analogique pour le potentiomètre de l'axe Y
5	SW	Sortie de commutation. La broche devient Bas (O V) lorsque l'interrupteur est enfoncé

Vous avez besoin de deux broches d'entrée analogiques et d'une broche d'entrée numérique sur l'Arduino pour terminer la connexion.

Vous pouvez utiliser l'une des deux entrées analogiques de l'Arduino UNO.

Arduino UNO dispose de six broches d'entrée analogiques étiquetées A0, A1, A2, A3, A4 et A5.

Bien que j'aie dit que l'ADC lirait 512 points lorsque le potentiomètre est inactif, ce ne sera pas précisément à mi-chemin. Vous pouvez voir un décalage de 5 ou 10 valeurs. Par conséquent, vous pouvez calibrer les trois positions : extrême gauche, extrême droite et ralenti.

Lors du calibrage du joystick, prenez suffisamment de valeurs et faites-en la moyenne pour trouver le décalage.

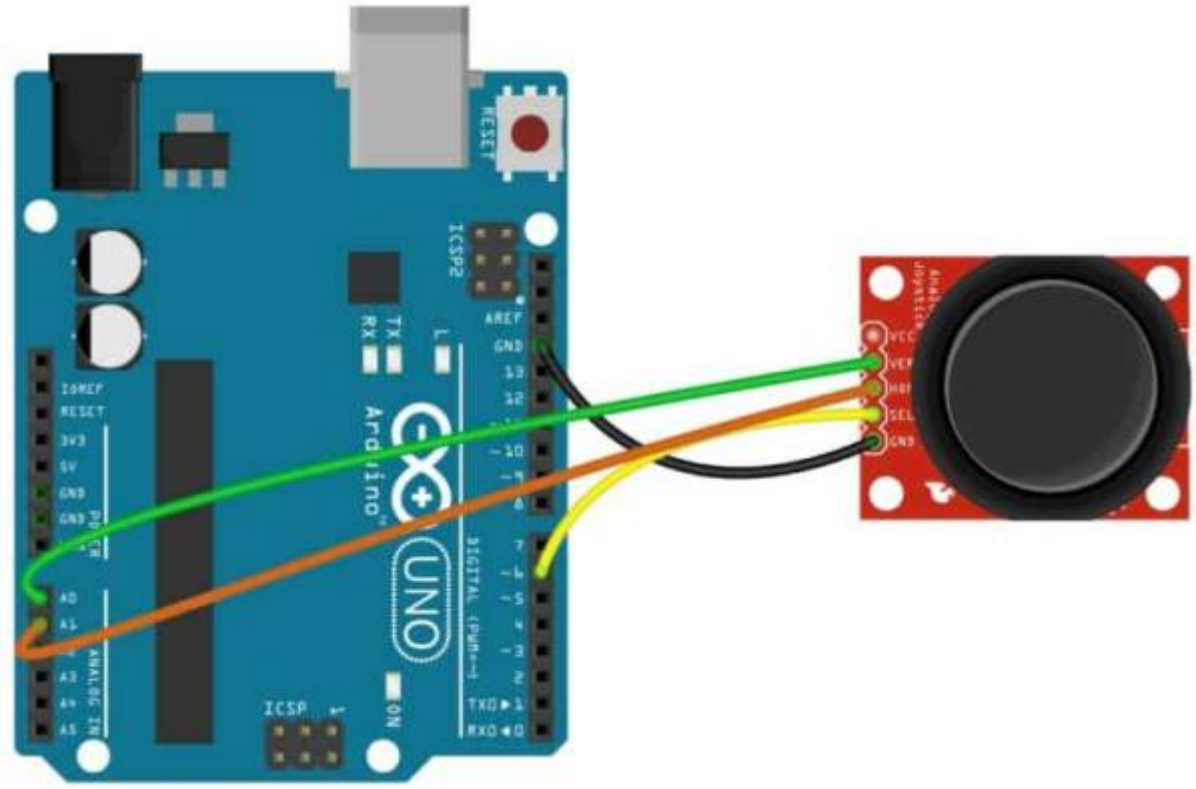
Je recommande de prendre au moins 100 valeurs. Cela permet de supprimer le bruit et de choisir une meilleure valeur de correction de décalage.

Voyons l'étalonnage du potentiomètre de l'axe X.

- Gardez le joystick au ralenti (position centrale)
- Lisez la valeur analogique de l'ADC.
- Assurez-vous que la lecture est comprise entre 500 et 520. Ceci permet de garantir que la position de l'arbre est dans la position/l'état attendu.
- Continuez à ajouter les lectures à une variable.
- Répétez les étapes de 2 à 4 100 fois
- Calculer la moyenne (Somme totale / 100)
- Stockez la nouvelle valeur « moyenne ».

De même, vous pouvez calibrer le potentiomètre de l'axe X pour les positions finales. Suivez également les mêmes étapes pour le potentiomètre de l’axe Y.

Comment connecter le module joystick avec Arduino UNO



-1- Connectez l'une des broches GND de l'Arduino à la broche étiquetée « GND » sur le module joystick.

J'insiste toujours pour démarrer les connexions en connectant d'abord les connexions GND.

-2- La broche SEL peut également être étiquetée SW. Il s'agit de l'interrupteur momentané présent dans le module.

N'oubliez pas d'activer le pullup interne sur cette broche. Connectez la broche SEL à la broche 6 de l'Arduino UNO.

-3- Connectez la broche d'entrée analogique A1 de l'Arduino UNO à la broche marquée « HOR » du module. L'étiquette HOR est également connue sous le nom d'axe X/VRx,

-4- Connectez la broche analogique A0

Connectez la broche d'entrée analogique A0 de l'Arduino UNO à la broche étiquetée « VER » du module. L'étiquette HOR est également connue sous le nom d'axe Y/VRy,

-5- Connectez la broche du capteur étiquetée VCC à la broche Arduino 5 V.

Programme Arduino contrôle servo avec joystick

joystick001.ino

```
#define pinX A1

#include "Servo.h"
```

```
Servo servo1;

void setup() {
   pinMode(pinX, INPUT);

   servo1.attach(12);
}

void loop() {
   int X = analogRead(pinX);

   X = map(X, 0, 1023, 0, 180);

   servo1.write(X);
}
```

Utilisation de la manette Joystick avec Arduino

joystick002.ino

```
#include <LiquidCrystal_I2C.h>
//—— Adressage matériel ——
// En cas de non fonctionnement, mettez la ligne 8 en
// commentaire et retirez le commentaire à la ligne 9.
LiquidCrystal_I2C lcd(0x27, 20, 4);

int VRx = A0;
int VRy = A1;
int SW = 2;

int xPosition = 0;
int yPosition = 0;
int SW_state = 0;
int mapX = 0;
int mapY = 0;

void setup() {
lcd.init(); // initialisation de l'afficheur
pinMode(VRx, INPUT);
pinMode(VRy, INPUT);
pinMode(SW, INPUT_PULLUP);

}

void loop() {
lcd.backlight();
lcd.clear(); // effacer le contenu de l'Afficheur LCD
xPosition = analogRead(VRx);
```

```
yPosition = analogRead(VRy);
SW_state = digitalRead(SW);
mapX = map(xPosition, 0, 1023, -512, 512);
mapY = map(yPosition, 0, 1023, -512, 512);

if ((mapX>=-515)&& (mapX<=-510)&&(mapY>=-175)&& (mapY<=-168))
lcd.print("Avant");
if ((mapX>=168)&& (mapX<=175)&&(mapY>=-175)&& (mapY<=-168))
lcd.print("Arriere");
if ((mapX<=-166)&& (mapX>=-175)&&(mapY>=-515)&& (mapY<=-510))
lcd.print("Droite");
if ((mapX>=-175)&& (mapX<=-170)&&(mapY>=165)&& (mapY<=175))
lcd.print("Gauche");
if (SW_state==0)
lcd.print("Bouton appuye");
delay(1000);
}
```

Contrôleur de joystick pour robot MeArm

joystick003.ino

```
/* meArm analog joysticks version 1.3.1 - UtilStudio.com Dec 2018
   Uses two analogue joysticks and four servos.

   In version 1.3 was improved  recording of coordinates.
   Some bugs was removed.

   First joystick  moves gripper forwards, backwards, left and right,
   button start/stop recording  positions.

   Second joystick moves gripper up, down, and closes and opens,
   button start/stop playing recorded positions.
   Press button for 2 seconds  to autoplay.

   Pins:
   Arduino    Stick1    Stick2    Base   Shoulder  Elbow    Gripper
Record/
      GND       GND       GND    Brown     Brown   Brown     Brown
Auto play
       5V       VCC       VCC      Red       Red     Red       Red
LED
       A0       HOR
       A1       VER
       PD2      BUTT
       A2                 HOR
       A3                 VER
       PD3                BUTT
```

```
       11                        Yellow
       10                                 Yellow
        9                                          Yellow
        6                                                   Yellow
       PD4                                                          X
*/
#include  <Servo.h>

bool repeatePlaying = false; /* Repeatedly is running recorded  cycle
*/
int delayBetweenCycles = 2000; /* Delay between cycles */

int  basePin = 11;       /* Base servo */
int shoulderPin = 10;   /* Shoulder servo  */
int elbowPin = 9;       /* Elbow servo */
int gripperPin = 6;     /* Gripper  servo */

int xdirPin = 0;        /* Base - joystick1*/
int ydirPin = 1;        /* Shoulder - joystick1 */
int zdirPin = 3;        /* Elbow - joystick2  */
int gdirPin = 2;        /* Gripper - joystick2 */

//int pinRecord =  A4;     /* Button record - backward compatibility */
//int pinPlay = A5;       /*  Button play  - backward compatibility */
int pinRecord = PD2;     /* Button record  - recommended (A4 is
deprecated, will by used for additional joystick) */
int  pinPlay = PD3;       /* Button play  - recommended (A5 is
deprecated, will by used  for additional joystick) */
int pinLedRecord = PD4;  /* LED - indicates recording  (light) or auto
play mode (blink one) */

bool useInternalPullUpResistors  = false;

const int buffSize = 512; /* Size of recording buffer */

int  startBase = 90;
int startShoulder = 90;
int startElbow = 90;
int startGripper  = 0;

int posBase = 90;
int posShoulder = 90;
int posElbow = 90;
int  posGripper = 0;

int lastBase = 90;
int lastShoulder = 90;
int lastElbow  = 90;
int lastGripper = 90;

int minBase = 0;
```

```
int maxBase = 150;
int  minShoulder = 0;
int maxShoulder = 150;
int minElbow = 0;
int maxElbow  = 150;
int minGripper = 0;
int maxGripper = 150;

const int countServo  = 4;
int buff[buffSize];
int buffAdd[countServo];
int recPos = 0;
int  playPos = 0;

int buttonRecord = HIGH;
int buttonPlay = HIGH;

int  buttonRecordLast = LOW;
int buttonPlayLast = LOW;

bool record = false;
bool  play = false;
bool debug = false;

String command = "Manual";
int  printPos = 0;

int buttonPlayDelay = 20;
int buttonPlayCount = 0;

bool  ledLight = false;

Servo servoBase;
Servo servoShoulder;
Servo servoElbow;
Servo  servoGripper;

void setup() {
  Serial.begin(9600);

  if (useInternalPullUpResistors)  {
    pinMode(pinRecord, INPUT_PULLUP);
    pinMode(pinPlay, INPUT_PULLUP);
  }
  else
  {
    pinMode(pinRecord, INPUT);
    pinMode(pinPlay, INPUT);
  }

  pinMode(xdirPin, INPUT);
```

```
  pinMode(ydirPin, INPUT);
  pinMode(zdirPin,  INPUT);
  pinMode(gdirPin, INPUT);

  pinMode(pinLedRecord, OUTPUT);

  servoBase.attach(basePin);
  servoShoulder.attach(shoulderPin);
  servoElbow.attach(elbowPin);
  servoGripper.attach(gripperPin);

  StartPosition();

  digitalWrite(pinLedRecord,  HIGH);
  delay(1000);
  digitalWrite(pinLedRecord, LOW);
}

void  loop() {

  buttonRecord = digitalRead(pinRecord);
  buttonPlay = digitalRead(pinPlay);

  //  Serial.print(buttonRecord);
  //  Serial.print(" ");
  //  Serial.println(buttonPlay);
  //  for testing purposes

  if (buttonPlay == LOW)
  {
    buttonPlayCount++;

    if (buttonPlayCount >= buttonPlayDelay)
    {
      repeatePlaying = true;
    }
  }
  else buttonPlayCount = 0;

  if (buttonPlay != buttonPlayLast)
  {
    if (record)
    {
      record = false;
    }

    if  (buttonPlay == LOW)
    {
      play = !play;
      repeatePlaying = false;
```

```
    if (play)
    {
      StartPosition();
    }
  }
}

if (buttonRecord != buttonRecordLast)
{
  if (buttonRecord == LOW)
  {
    record = !record;

    if (record)
    {
      play  = false;
      repeatePlaying = false;
      recPos = 0;
    }
    else
    {
      if (debug) PrintBuffer();
    }
  }
}

buttonPlayLast = buttonPlay;
buttonRecordLast = buttonRecord;

float dx = map(analogRead(xdirPin), 0, 1023, -5.0, 5.0);
float dy = map(analogRead(ydirPin),  0, 1023, 5.0, -5.0);
float dz = map(analogRead(zdirPin), 0, 1023, 5.0, -5.0);
float dg = map(analogRead(gdirPin), 0, 1023, 5.0, -5.0);

if (abs(dx)  < 1.5) dx = 0;
if (abs(dy) < 1.5) dy = 0;
if (abs(dz) < 1.5) dz = 0;
if (abs(dg) < 1.5) dg = 0;

posBase += dx;
posShoulder += dy;
posElbow += dz;
posGripper += dg;

if (play)
{
  if (playPos  >= recPos) {
    playPos = 0;

    if (repeatePlaying)
    {
```

```
      delay(delayBetweenCycles);
      StartPosition();
    }
    else
    {
      play = false;
    }
  }

  bool endOfData =  false;

  while (!endOfData)
  {
    if (playPos >= buffSize -  1) break;
    if (playPos >= recPos) break;

    int data = buff[playPos];
    int angle = data & 0xFFF;
    int servoNumber = data & 0x3000;
    endOfData  = data & 0x4000;

    switch (servoNumber)
    {
      case 0x0000:
        posBase = angle;
        break;

      case 0x1000:
        posShoulder  = angle;
        break;

      case 0x2000:
        posElbow =  angle;
        break;

      case 0x3000:
        posGripper =  angle;
        dg = posGripper - lastGripper;
        break;
    }

    playPos++;
  }
}

if (posBase > maxBase) posBase = maxBase;
if (posShoulder > maxShoulder) posShoulder = maxShoulder;
if (posElbow >  maxElbow) posElbow = maxElbow;
if (posGripper > maxGripper) posGripper = maxGripper;

if (posBase < minBase) posBase = minBase;
```

```
  if (posShoulder < minShoulder)  posShoulder = minShoulder;
  if (posElbow < minElbow) posElbow = minElbow;
  if (posGripper < minGripper) posGripper = minGripper;

  servoBase.write(posBase);
  servoShoulder.write(posShoulder);
  servoElbow.write(posElbow);

  bool  waitGripper = false;
  if (dg < 0) {
    posGripper = minGripper;
    waitGripper  = true;
  }
  else if (dg > 0) {
    posGripper = maxGripper;
    waitGripper  = true;
  }

  servoGripper.write(posGripper);
  if (play && waitGripper)
  {
    delay(1000);
  }

  if ((lastBase != posBase) | (lastShoulder  != posShoulder) |
(lastElbow != posElbow) | (lastGripper != posGripper))
  {
    if (record)
    {
      if (recPos < buffSize - countServo)
      {
        int buffPos = 0;

        if (lastBase != posBase)
        {
          buffAdd[buffPos] = posBase;
          buffPos++;
        }

        if (lastShoulder != posShoulder)
        {
          buffAdd[buffPos]  = posShoulder | 0x1000;
          buffPos++;
        }

        if (lastElbow  != posElbow)
        {
          buffAdd[buffPos] = posElbow | 0x2000;
          buffPos++;
        }
```

```
        if (lastGripper != posGripper)
        {
          buffAdd[buffPos] = posGripper | 0x3000;
          buffPos++;
        }

        buffAdd[buffPos - 1] = buffAdd[buffPos - 1] | 0x4000;

        for (int i = 0; i < buffPos; i++)
        {
          buff[recPos  + i] = buffAdd[i];
        }

        recPos += buffPos;
      }
    }

    command = "Manual";
    printPos = 0;

    if (play)
    {
      command = "Play";
      printPos = playPos;
    }
    else  if (record)
    {
      command = "Record";
      printPos = recPos;
    }

    Serial.print(command);
    Serial.print(" ");
    Serial.print(printPos);
    Serial.print(" ");
    Serial.print(posBase);
    Serial.print(" ");
    Serial.print(posShoulder);
    Serial.print(" ");
    Serial.print(posElbow);
    Serial.print(" ");
    Serial.print(posGripper);
    Serial.print(" ");
    Serial.print(record);
    Serial.print(" ");
    Serial.print(play);
    Serial.println();
  }

  lastBase = posBase;
  lastShoulder = posShoulder;
```

```
  lastElbow = posElbow;
  lastGripper = posGripper;

  if ( repeatePlaying)
  {
    ledLight = !ledLight;
  }
  else
  {
    if (ledLight)
    {
      ledLight = false;
    }

    if (record)
    {
      ledLight  = true;
    }
  };

  digitalWrite(pinLedRecord, ledLight);
  delay(50);
}

void  PrintBuffer()
{
  for (int i = 0; i < recPos; i++)
  {
    int data  = buff[i];
    int angle = data & 0xFFF;
    int servoNumber = data & 0x3000;
    bool endOfData = data & 0x4000;

    Serial.print("Servo=");
    Serial.print(servoNumber);
    Serial.print(" Angle=");
    Serial.print(angle);
    Serial.print(" End=");
    Serial.print(endOfData);
    Serial.print(" Data=");
    Serial.print(data,  BIN);
    Serial.println();
  }
}

void StartPosition()
{
  int angleBase = servoBase.read();
  int angleShoulder = servoShoulder.read();
  int angleElbow = servoElbow.read();
  int angleGripper = servoGripper.read();
```

```
  Serial.print(angleBase);
  Serial.print("   ");
  Serial.print(angleShoulder);
  Serial.print("   ");
  Serial.print(angleElbow);
  Serial.print("   ");
  Serial.print(angleGripper);
  Serial.println(" ");

  posBase = startBase;
  posShoulder = startShoulder;
  posElbow = startElbow;
  posGripper = startGripper;

  servoBase.write(posBase);
  servoShoulder.write(posShoulder);
  servoElbow.write(posElbow);
  servoGripper.write(posGripper);
}
```