TRABAJO DE FIN DE GRADO.

Grado en Ingeniería Electrónica Industrial y Automática.

Construcción de una plataforma robótica para la ejecución coordinada de movimientos de múltiples manipuladores.

Miguel Ángel Cobas Ortiz.

Héctor Castro Estévez.

La laguna, 7 de Septiembre de 2017.

D. **Santiago Torres Álvarez**, con N.I.F. 43369478-B, profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y Sistemas de la Universidad de La Laguna, como tutor

CERTIFICA

Que la presente memoria titulada:

"Construcción de una plataforma robótica para la ejecución coordinada de movimientos de múltiples manipuladores"

ha sido realizada bajo su dirección por D. **Miguel Ángel Cobas Ortiz**, con N.I.F. 42223397-C y por D. **Héctor Castro Estévez** con N.I.F. 78633200-H.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firma la presente en La Laguna a 7 de septiembre de 2017.

Índice.

1. Definiciones.
2. Abstract.
3. Introducción.
4. Fabricación de la plataforma robótica.
   - 4.1. Plataforma y tacos elevadores.
   - 4.2. Placas de aluminio.
   - 4.3. Placas de metacrilato.
   - 4.4. Montaje de componentes.
5. Cálculos.
   - 5.1. Cambio de sistemas de referencia.
   - 5.2. Cinemática directa de un manipulador R-R.
   - 5.3. Cinemática inversa de un manipulador R-R.
6. Diseño de programa.
7. Desarrollo del prototipo, imprevistos y soluciones adoptadas.
8. Conclusiones.
9. Conclusions.
10. Posibles mejoras.
11. Bibliografía.
    - 11.1. Aplicaciones y programas utilizados.
12. Presupuesto.
13. Anexos.
    - 13.1. Anexo I. Código.
      - 13.1.1. Cinemática directa.
      - 13.1.2. Cinemática inversa.
    - 13.2. Anexo II. Datasheet Servomotor.
    - 13.3. Anexo III. Placa Arduino UNO.
    - 13.4. Anexo IV. Especificaciones de la carcasa exterior.
    - 13.5. Anexo V. Módulo adaptador de micro-SD.

1. Definiciones.

- Servomotor o servo: motor auxiliar para hacer que gire el manipulador. Como en cada manipulador hay dos servomotores, se pueden distinguir como:
    - Servomotor principal: ubicado de manera fija en el exterior de la plataforma.
    - Servomotor secundario: ubicado entre el servomotor principal y el efector final. No es fijo dado que su posición varía en relación al ángulo del servomotor principal.
- Placa: elemento de conexión del manipulador. Como en cada manipulador hay 2 placas, se pueden distinguir como:
    - Placa principal: Hecha de aluminio. Es la que une el servomotor principal y el secundario. Su orientación varía en relación al ángulo del servomotor principal.
    - Placa secundaria: Hecha de metacrilato. Es la que conecta el servomotor secundario con el efector final. Su orientación varía en relación al ángulo del servomotor secundario.
- Manipulador: Brazo robótico que, en nuestro caso, consta de dos grados de libertad. Cada una de las articulaciones es de revolución (por lo que el manipulador será de tipo R-R) y está compuesta por dos servomotores y dos placas.
- Efector final: extremo del manipulador sobre el que se harán los cálculos cinemáticos directo e inverso del manipulador. Concretamente, se escogerá el punto medio del final de la placa secundaria para la realización de dichos cálculos por parte de la programación.
- Tacos elevadores:Paralelepípedo recto de madera colocado bajo el servomotor principal, cuya función principal es elevar el manipulador para evitar colisión con la plataforma debido a la holgura de las placas.
- Plataforma: Es la superficie circular sobre la que se montan los cuatro manipuladores.
- Carcasa: Es la fijación del servomotor.
- Acomodadores: Pieza negra que forma parte de los servomotores y cuya función principal es fijación de las piezas que se quieran dotar de movimiento mediante el giro del servomotor, en nuestro caso las placas principal y secundaria de cada manipulador.

2. Abstract.

The aim of this project is to carry out the manufacture of a circular platform with four robotic2 DOF[1]manipulators (in the R-R configuration), as well as the programming and the implementation with an Arduino board.

The main objective of this work is the elaboration of a prototype for developing and testing advanced control algorithms, that will be implemented in the astronomical instrument MIRADAS (Mid-resolutionInfRAreDAstronomical Spectrograph). MIRADAS is a project currently in progress by a consortium of companies and institutions including the InstitutoAstrofísico de Canarias (IAC), and the resultant instrument which will be located at the Gran Telescopio de Canarias (GTC), located in the Roque de losMuchachos observatory, in the island of La Palma. Its objective is to use multiple robotic manipulators, coupled on the inner surface of a telescope, to adjust the position of the mirrors located at the edge of the arms, so that multiple objectives could be measured simultaneously. For testing the control algorithms for the manipulators movements, in order to avoid collisions of the robots in the trajectories towards their respective objectives, it is necessary to use a scaled prototype for programming, control and simulate different control situations.The designed control algorithms can be then tested and later used in the final design of the instrument controller.

There are three goals to achieve:

- To program the manipulators movement using the direct kinematics based on position.

- To program the manipulators movement using the inverse kinematics based on position.

-To program the manipulators movement using the direct kinematics based on speed.

The required conditions referred with the prototype are:

- The final point of each manipulator must exceed the center of the platform, being able to collide with other manipulators, with the purpose of getting larger movement's range of the mirrors in the telescope.

- The main link of the different manipulators cannot collide between them. The collisions could occur between the secondary links.

- The data must be received in ".txt" format.

[1] DOF is the acronym of Degree of Freedom.

3. Introducción.

En este proyecto se pretende llevar a cabo la fabricación de una plataforma circular con cuatro manipuladores robóticos de dos grados de libertad cada uno (en configuración R-R), así como su programación e implementación en una placa Arduino.

El objetivo principal de este trabajo consiste en la elaboración de un prototipo para la aplicación de algoritmos de control avanzados que serán implementados a posteriori en el instrumento astronómico MIRADAS (*Mid-resolutionInfRAreDAstronomicalSpectrograph*), proyecto actualmente en curso por parte de un consorcio de empresas e instituciones entre las cuales está el Instituto Astrofísico de Canarias (IAC) para ser ubicado en el Gran Telescopio de Canarias (GTC), ubicado en el observatorio del Roque de los Muchachos en la isla de La Palma. El objetivo de dicho proyecto es usar múltiples manipuladores robóticos, acoplados en la superficie interior de un telescopio, para ajustar la posición de los espejos ubicados en el extremo de dichos brazos, de forma que se puedan medir múltiples objetivos a la vez. Para la validación de los algoritmos de control del movimiento de los manipuladores, con el fin de evitar colisiones de los robots en las trayectorias hacia sus respectivos objetivos, se necesita un prototipo a escala para poder programar, controlar y simular situaciones de control diversas, con las cuales se pueda validar los algoritmos de control diseñados, y que posteriormente se utilizarán en el diseño final del controlador del instrumento.

Los tres hitos a conseguir en el proyecto son:

- Programar el movimiento de los manipuladores mediante el uso de la cinemática directa en posición.
- Programar el movimiento de los manipuladores mediante el uso de la cinemática inversa en posición.
- Programar el movimiento de los manipuladores mediante el uso de la cinemática directa en velocidad.

Las condiciones a la hora de realizar el diseño del prototipo son:

- Que el punto final de cada uno de los manipuladores sobrepase el centro de la plataforma, pudiendo colisionar entre sí, con la finalidad de poder conseguir mayor rango a la hora de mover los espejos en el telescopio.
- Que no pueda colisionar la placa principal de un manipulador con la placa principal de otro. Las colisiones se darán entre las placas secundarias de los manipuladores.
- Recepción de datos a través de archivo *".txt".*

4. Fabricación de la plataforma robótica.

La idea del proyecto consiste en fabricar una plataforma plana, sobre la que van a ir fijados cuatro servomotores ubicados de manera uniforme en el extremo de dicha plataforma. Encima de cada uno de esos cuatro servomotores se debe colocar una placa rectangular, en cuyo extremo se colocaría otro servomotor con otra placa, cuyo límite simula el extremo del manipulador. También es importante recordar la conexión mediante cables de la placa Arduino, así como la fuente de alimentación, por lo que se ha de elevar dicha plataforma para colocar la placa debajo de esta y llevar hasta ella los cables de los servomotores. Por otro lado, es importante resaltar que los manipuladores, en nuestro diseño, pueden colisionar entre sí, lo cual implica que a la hora de programar habrá que tener cuidado para que no choquen los brazos robóticos, desajustando los servomotores y generando errores de precisión.

Cabe destacar que, al poner los servomotores por la misma cara de la placa principal, la placa secundaria hará contacto con la plataforma. Para solucionar dicho problema, se recurre a la adición de unos tacos elevadores-con la misma superficie que la carcasa del servomotor- entre la plataforma y el primer servo. El esquema final se puede apreciar en la *figura 1*.

Las placas principales utilizadas serán de aluminio para evitar el pandeo que se crea debido al peso, y las placas secundarias serán de metacrilato ya que pesan menos y es más económico.

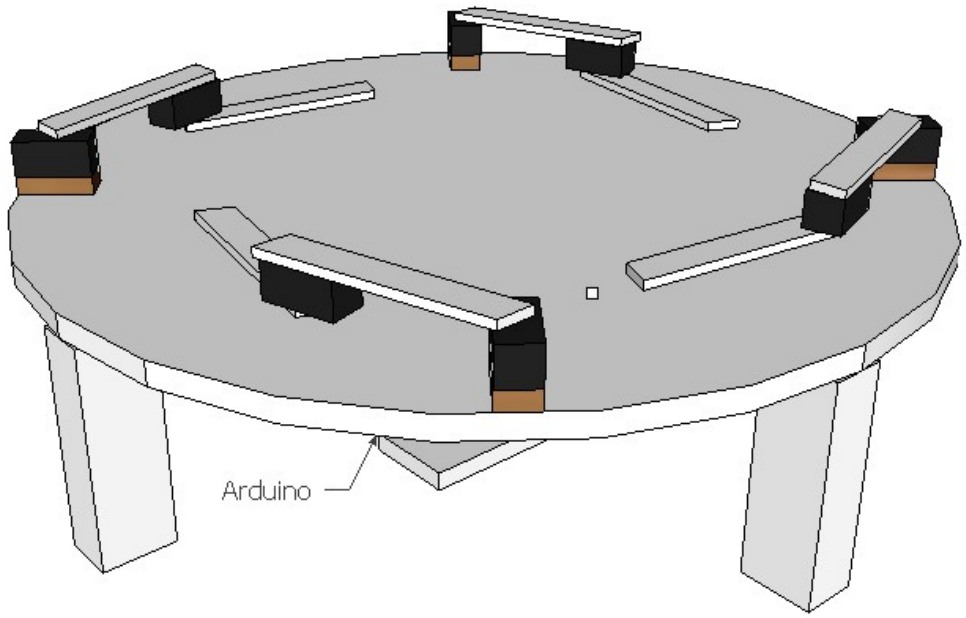


Figura 1. Esquema del montaje.

4.1.Plataforma y tacos elevadores.

A la hora de fabricar la plataforma y los tacos elevadores,se opta por un tablero de madera de 1 cm. de grosor para la plataforma y una madera de 2 cm. de grosor para los tacos, dado que la plataforma y los tacos han de estar adheridos mediante tornillos y la elevación de los tacos ha de ser suficientemente alta para que las placas no colisionen con la plataforma.

Las medidas de la plataforma deben ajustarse al tamaño de los servomotores y de las placas del manipulador. Teniendo esto en cuenta, se decide diseñar la plataforma con un radio de 25 cm. x 1 cm. Para la medida de los tacos elevadores simplemente se utiliza la misma superficie de la carcasa del servomotor, con un grosor de 2cm. de elevación, es decir,4,5 cm. x 2,8 cm. x 2 cm.

Para obtener la forma de la plataforma y de los tacos, se utiliza una caladora para cortar el tablero de la plataforma y una tronzadora para cortar la madera de los tacos (ya que esta era muy gruesa). Después, se le da un acabado conla lija y se agujerean los tacos para facilitar luego la introducción de tornillos. Todo lo realizado ha sidoutilizando el equipo de protección individual correspondiente y sujetando la pieza con una sargenta, en las *figuras 2, 3, 4, 5, 6 y 7* se puede ver el proceso.

Figura 2. Cortando plataforma.

Figura 3. Tacos fase 1.

Figura 4. Tacos fase 2.

Figura 5. Tacos fase 3.

Figura 6. Acabado de los tacos.

Figura 7. Fijación de tacos en plataforma.

4.2.Placas de aluminio.

Para las medidas de la placa principal hay que tener en cuenta que no pueden colisionar dos placas principales de distintos manipuladores, por lo que la medida máxima para dicha placa principal será:

$$(1)\frac{h}{2} = \sqrt{25^2 + 25^2} \cdot 0{,}5 = 17{,}68\ cm$$

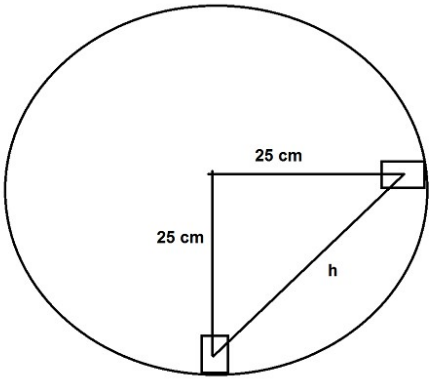


Figura 8. Cálculo de la plataforma principal.

Entonces, la distancia de cada placa debe ser como máximo de 17,68 cm., pero para evitar rozamientos debido al ancho de la placa se eligen 16cm. Como la placa tiene que fijarse mediante tornillos, aumentamos la distancia 2,5cm. por el lado del servomotor principal para poder perforar la placa sin problema. Finalmente, la medida tendrá como dimensiones 18,5 cm. x 5 cm. x 0,3 cm.

Para dar forma al aluminio, se utilizó una caladora con sierra para metales con los equipos de protección individual correspondiente, es decir, guantes y gafas de protección, sujetando la pieza con una sargenta.

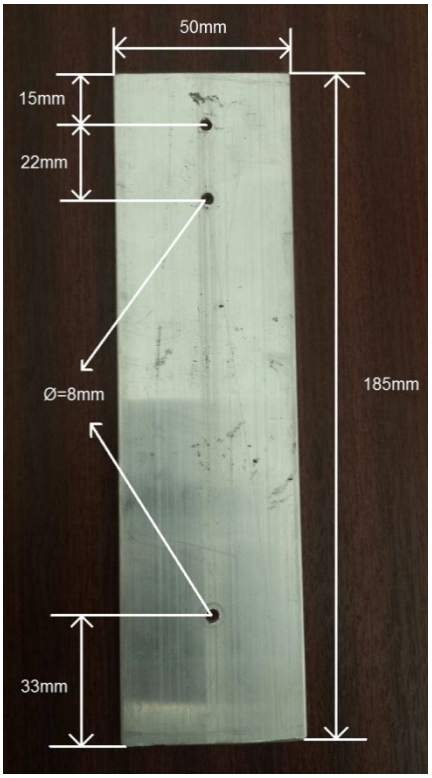


Figura 9. Ilustración y medidas de las placas principales.

4.3.Placas de metacrilato

Para fabricar las placas de metacrilato, se eligen medidas que no sean muy gruesas para que no suponga un problema en cuanto a la colisión, por lo que serán un poco más anchas que los servomotores. Ahora, como se debe superar la distancia del centro de la plataforma, la distancia a elegir deberá ser como mínimo:

$$R - D_0 - L_{PP} = L_{PS}$$

25 – 1,2 -11,5 = 12,5 cm. = L_{PS}

R: Radio de la plataforma.

D_0: distancia desde el borde de la plataforma hasta el rotor del servomotor principal.

L_{PP}: distancia de la placa principal.

L_{PS}: longitud de la placa secundaria.

Finalmente, la dimensión elegida para el largo del rectángulo que representa la superficie de la placa es 12,5 cm., a eso se le añade un margen para poder atornillarla de 2,2 cm., obteniendo unas dimensiones finales de14,7 cm. x 3,5 cm. x 0,03 cm.

A la hora de dar forma al metacrilato, partiendo de una placa grande, se utiliza una tronzadora eléctrica con el correspondiente equipo de protección individual, guantes y gafas de protección. Asimismo, hay que destacar que hay que perforar la placa para sujetarla al servomotor secundario, para lo cual se utiliza un taladro de broca 10 Ømm. En la *figura 10* se muestra el acabado de las placas de metacrilato:

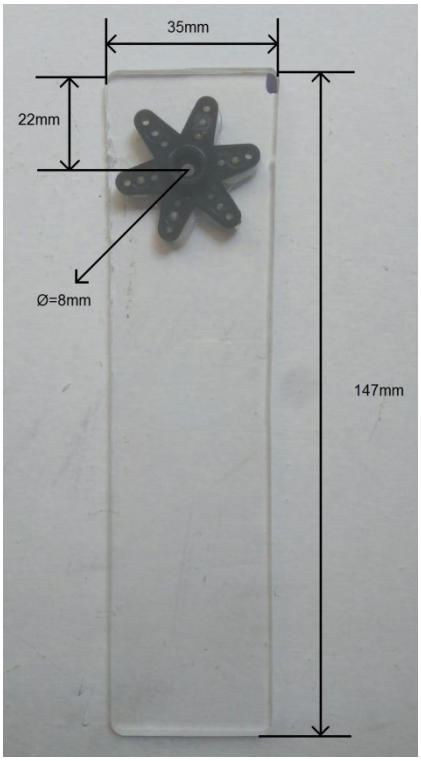


Figura 10. Ilustración y medidas de las placas secundarias.

4.4.Montaje de componentes

En cuanto al montaje de los componentes, para su correcto funcionamiento primero se ha de taladrar el taco a la plataforma, para así poner encima la carcasa del servomotor principal con su respectivo servo (fijado mediante tornillos, arandelas y tuercas), de tal manera que la orientación de este nos facilite la salida de los cables al exterior de la plataforma, ya que los cables van conectados a la placa Arduino. Por otro lado, se fija el servomotor secundario con su respectiva carcasa y se atornilla la placa secundaria por la parte del efector del servomotor (orientado también para facilitar la salida de cables al exterior de la plataforma). Asimismo, se ha de atornillar la parte inferior de la carcasa del servomotor secundario a la placa principal. Finalmente se atornilla la placa principal, ya unida al servomotor secundario con su carcasa y la placa secundaria, al servomotor principal.

En cuando a la elevación de la plataforma utilizamos una estructura metálica que cumpla dicha función, dicha estructura se puede apreciar en la *figura 11.*

Figura 11. Estructura metálica de sujeción de la plataforma.

Por otra parte, hay que realizar las conexiones entre la placa Arduino, el módulo adaptador de micro SD, la fuente de alimentación y los servomotores, las cuales se pueden apreciar en la*figura 12.*

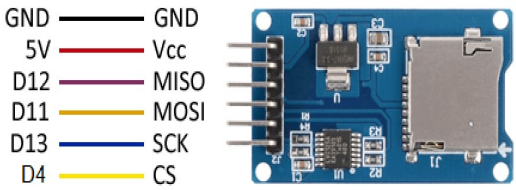


Figura 12. Módulo adaptador de micro-SD.

Los cables de los servomotores son tres, uno de color rojo conectado a la alimentación (de 4,8 V a 6 V), otro marrón conectado a la tierra (la cual es común de la placa Arduino y la fuente

de alimentación), y el último de color amarillo conectado a la señal (cada servomotor conectado a un pin de salida de la Arduino).

La utilización de la fuente de alimentación es necesaria ya que, debido a la intensidad tan baja que suministra la placa Arduino, no se pueden mover más de tres servomotores conectados sólo a esta.

El resultado final es el que se muestra en la *figura 13*. Asimismo, en la *figura 14* se puede apreciar un esquema del cableado de los distintos componentes de la plataforma robótica.

Figura 13. Montaje final de la plataforma robótica.

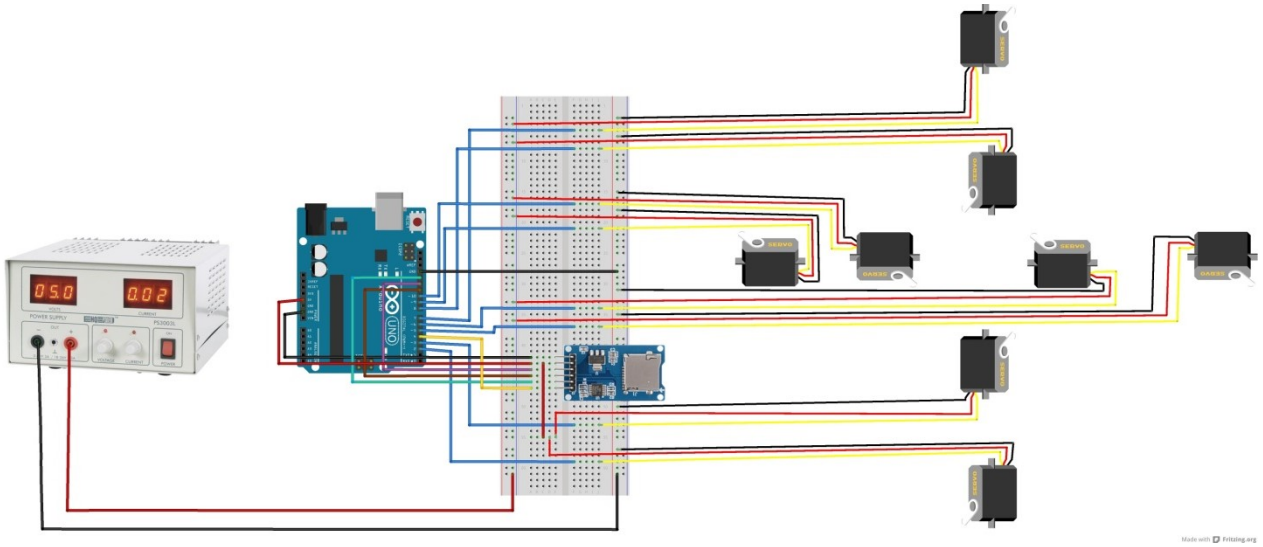


Figura 14. Esquema del cableado implementado en el montaje.

5. Cálculos.

En este apartado se recogen los cálculos necesarios para el movimiento de los servomotores, de forma que la estructura adquiera la configuración deseada. Estos cálculos se implementarán a continuación en la programación del Arduino, de forma que el usuario pueda directamente dar la configuración final deseada para la estructura en cada momento.

Los cálculos realizados se basan en la implementación de la cinemática directa e inversa, de tal modo que las coordenadas utilizadas en ambos sean en base a un sistema de referencia único, ubicado en el centro de la plataforma.

La cinemática directa consiste enla obtención de la posición en la que se encuentra el efector final después de asignarle unos determinados ángulos a los servomotores. Es decir, para cada manipulador obtenemos un punto en el sistema de coordenadas a partir de dos ángulos (uno para cada servomotor).

En cuanto a la cinemática inversa, es justamente lo contrario. Obtenemos, por cada manipulador, dos ángulos (uno para cada servomotor) a partir del punto final a alcanzar por el extremo del manipulador.

5.1.Cambio de sistemas de referencia.

La idea es que el usuario utilice un único sistema de coordenadas ubicado en el centro de la plataforma. Para ello, habrá que hacer un cambio de sistema de referencia en cada manipulador.

A la hora de realizar el cambio en cada manipulador, se ha de trasladar el sistema de coordenadas de cada manipulador al sistema de coordenadas central, como se muestra en la *figura 15*.

En cinemática directa utilizamos los ángulos de cada servomotor para hallar la posición del efector final en el sistema de coordenadas de cada manipulador y luego cambiamos el sistema de referencia al central.

En cinemática inversa calculamos (con las mismas fórmulas) a partir de la posición del sistema de referencia central, la posición en coordenadas del efector final en el sistema de coordenadas de cada manipulador. Con esto se obtienen los ángulos de los servomotores.

Donde:

$(5) x_1 = x_0$

$(6)\ y_1 = R + y_0$

$(7) x_2 = y_0$

$(8)\ y_2 = R - x_0$

$(9) x_3 = -x_0$

$(10)\ y_3 = R - y_0$

$(11) x_4 = -y_0$

$(12)\ y_4 = R + x_0$

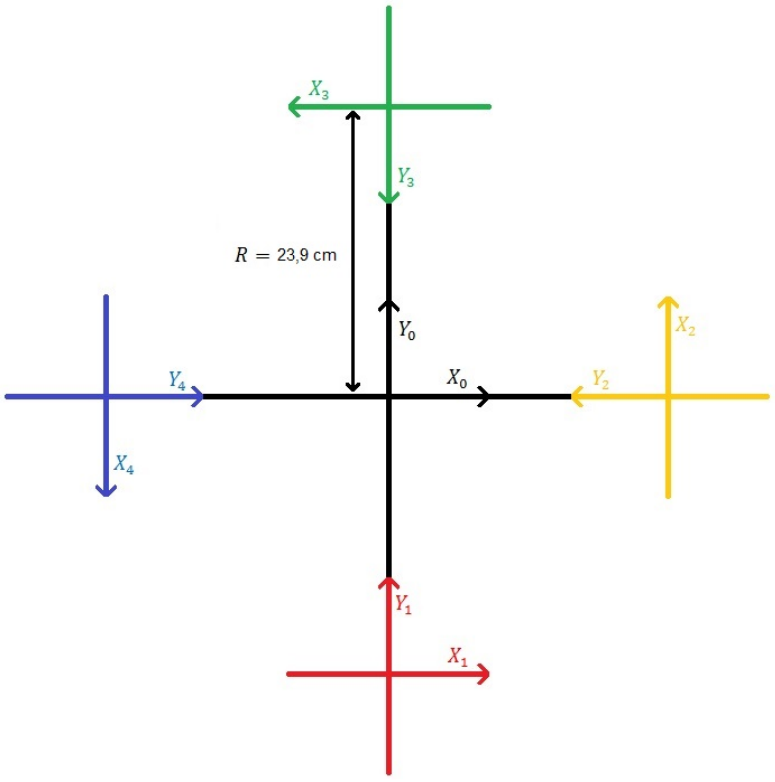


Figura 15. Esquema de coordenadas para el cambio de sistemas de referencia

5.2.Cinemática directa de un manipulador R-R.

Los cálculos en cinemática directa son tan sencillos como calcular, mediante trigonometría, las coordenadas del efector final, en base al sistema de coordenadas de cada servomotor. En la *figura 16* se pueden apreciar los valores de cada dato.

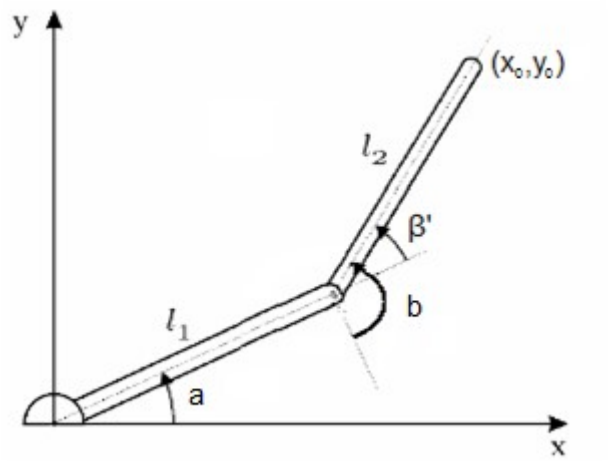


Figura 16. Planteamiento de la cinemática directa.

$$(2)\beta' = b - 90$$

$$(3)X_0 = l_1 \cdot \cos(a) + l_2 \cdot \cos(a + \beta')$$

$$(4)Y_0 = l_1 \cdot \sin(a) + l_2 \cdot sin(a + \beta')$$

5.3.Cinemática inversa de un manipulador R-R.

En cuanto a la cinemática inversa, hay que calcular los ángulos en función de la longitud de las placas, así como en función del punto de coordenadas que le indique el usuario. Para ello utilizamos el teorema del coseno para calcular los ángulos en configuración "codo abajo" y hacemos la resta pertinente al ángulo para hallar la configuración "codo arriba". En la *figura 17* se puede ver el dibujo del problema a plantear.

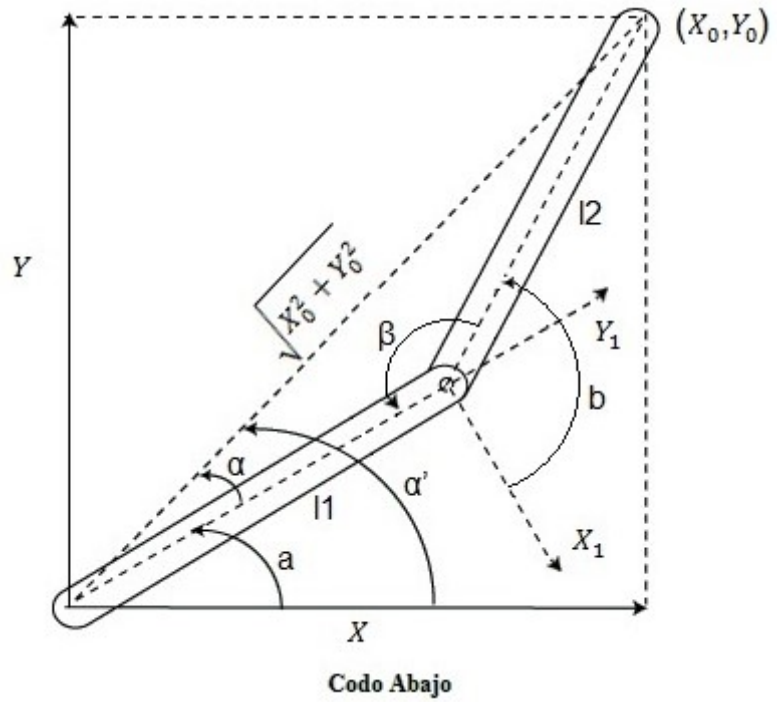


Figura 17. Planteamiento de la configuración "codo abajo".

Teniendo en cuenta que el manipulador está en la configuración "codo abajo". Las ecuaciones son:

$$(13)\quad \propto' = arccos\left[\frac{X_0}{\sqrt{X_0^2+Y_0^2}}\right]$$

$$(14)\quad \propto = arccos\left[\frac{-l_1^2+l_2^2+X_0^2+Y_0^2}{2*l_2*\sqrt{X_0^2+Y_0^2}}\right]$$

Donde el ángulo a:

$$(15)\quad a = \propto' - \propto$$

$$(16)\quad \beta = arccos\left[\frac{l_1^2+l_2^2-X_0^2-Y_0^2}{2*l_1*l_2}\right]$$

Y el ángulo b:

$$(17)\quad b = 270-\beta$$

Cabe destacar que las restas utilizadas en las ecuaciones de cálculo de (15) y (17), se utilizan para establecer el sistema de referencia requerido en función al eje de coordenadas deseado.

Para la configuración codo arriba, solo hay que aplicar las siguientes fórmulas, justificado en la *figura 18*, a los ángulos ya obtenidos:

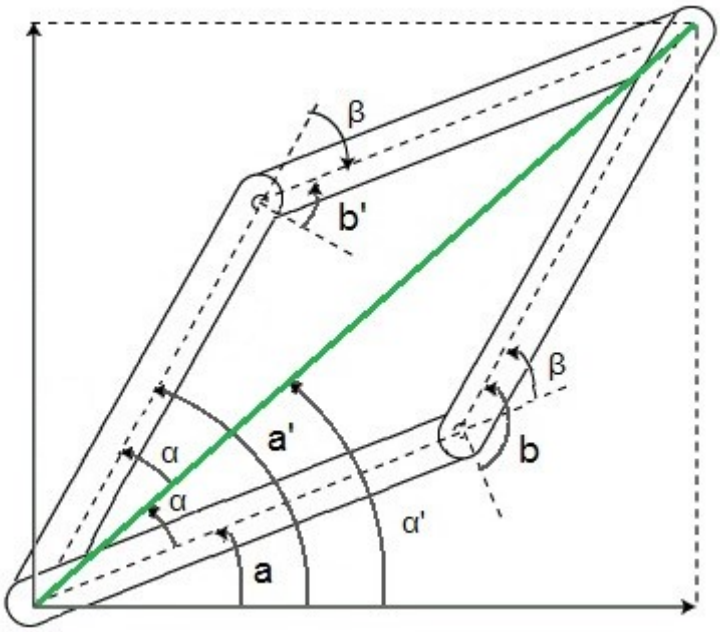


Figura 18. Planteamiento de la configuración codo arriba.

$$(18) \quad a' = \alpha + \alpha'$$

$$(19) b' = 90 + \beta$$

Donde a' y b' son los ángulos del manipulador con la configuración codo arriba.

Cabe mencionar que en la programación utilizamos la configuración "codo arriba"cuando las coordenadas del efector final están en los ejes "+X" y "+Y". En el caso de que los valores no sean válidos, por las limitaciones de los servomotores (desde 0^O a 210^O), analizamos la posibilidad de la configuración "codo abajo".

En el caso de que la posición del efector final se encuentre en los ejes "-X" y "-Y", se realiza el mismo proceso pero a la inversa (primero utilizamos la configuración "codo abajo" y si los valores no son válidos, "codo arriba").

6. Diseño del programa

Para el diseño del programa se utiliza el lenguaje de programación Arduino, el cual está basado en el lenguaje C. En este apartado se explicarán solo algunas partes de los códigos para comprender mejor su funcionamiento.

El extracto de código de la *figura 19* pertenece al programa de cinemática inversa. En la primera línea se elimina el archivo "nameFile.txt", para poder luego crearlo y escribir en él las coordenadas a las que se desee que lleguen los efectores finales de los manipuladores. Si se desea leer un archivo ya creado anteriormente, han de comentarse estas líneas. El programa de cinemática directa es prácticamente igual, a diferencia de los valores que han de añadirle al archivo, puesto que deben ser ángulos comprendidos entre el 0 y el 210.

```
SD.remove("nameFile.txt");

myFile = SD.open("nameFile.txt",FILE_WRITE);
if (myFile){
  myFile.print("(10,-5) (5,10) (-10,5) (-5,-10) .");
  myFile.close();
}else
{
  Serial.println(F("No se abrió correctamente"));
  myFile.close();
}
```

Figura 19.Creación de bloc de notas.

En el extracto de código que se observa en la *figura 20* se inicializan los servomotores a 0^{O}, habiéndose creado previamente un vector de servomotores llamado "servoMotor". Como los servomotores secundarios están colocados boca abajo, se deben invertir sus ángulos restándolos a 180, por ello se hace una distinción entre pares e impares, siendo los pares los servomotores principales y los impares los secundarios.

```
for(int i=0;i<8;i++){
  if(i%2==0){
    servoMotor[i].write(0);
  }else
  {
    servoMotor[i].write(180-0);
  }
}
```

Figura 20.Inicialización de los servomotores.

La manera en la que se obtienen valores numéricos, a partir de la cadena de caracteres existente dentro del bloc de notas, es leyendo carácter a carácter, guardando dichos números en una variable llamada “cadena” y, cuando se encuentre un espacio o una coma, convertir a “float” lo que se haya guardado y volcarlo en el primer espacio de un vector como se puede apreciar en la *figura 21*. Seguidamente se vacía la variable “cadena” y se avanza al siguiente espacio del vector. Esto termina cuando se reconoce un punto o un salto de línea.

```
while (myFile.available()) {
  char caracter=myFile.read();
  cadena=cadena+caracter;
  if(caracter=='(')
  {
    cadena="";
  }
  if(isSpace(caracter)==1 || caracter==','){
    var[j]= cadena.toFloat();
    Serial.print(F("var"));
    Serial.print(j);
    Serial.print(F(" es = "));
    Serial.println(var[j]);
    Serial.println(F("-----------------"));
    delay(100);
    j++;
    cadena="";
  }

  if(caracter==10 || caracter=='.' || caracter ==46)
  {
    break;
  }
}
```

Figura 21.Cambio de String a Float.

Como ya se explicó anteriormente, estas son las ecuaciones utilizadas para calcular la configuración codo arriba en la *figura 22* y la configuración codo abajo en la *figura 23* en el programa de cinemática inversa.

```
a[i]=((acos(x[i]/sqrt(x[i]*x[i]+y[i]*y[i])))+acos((-l1*l1+l2*l2+x[i]*x[i]+y[i]*y[i])/(2*l2*sqrt(x[i]*x[i]+y[i]*y[i]))))*RtD;
b[i]=((acos((l1*l1+l2*l2-x[i]*x[i]-y[i]*y[i])/(2*l1*l2)))*RtD)-90;
```

Figura 22.Ecuaciones utilizadas para la configuración codo arriba.

```
a[i]=((acos(x[i]/sqrt(x[i]*x[i]+y[i]*y[i])))-acos((-l1*l1+l2*l2+x[i]*x[i]+y[i]*y[i])/(2*l2*sqrt(x[i]*x[i]+y[i]*y[i]))))*RtD;
b[i]=270-((acos((l1*l1+l2*l2-x[i]*x[i]-y[i]*y[i])/(2*l1*l2)))*RtD);
```

Figura 23.Ecuaciones utilizadas para la configuración codo abajo.

En el extracto de código de la *figura 24,* perteneciente a la cinemática inversa, se asignan los valores de los ángulos “a[i]” y “b[i]”, calculados con las ecuaciones anteriores, multiplicados por “fit”, que es una constante utilizada para ajustar la imprecisión de los servomotores, al servomotor “k”. Siendo “i” el número de manipulador, “a” el ángulo del servomotor principal y “b” el del secundario, “k” el número del servomotor y “180 -” lo necesario para que el ángulo dado al secundario se corresponda con la realidad por estar colocado boca abajo.

```
servoMotor[k].write(a[i]*fit);
k++;
servoMotor[k].write(180-(b[i]*fit));
```

Figura 24.Asignación de ángulos en cinemática inversa.

En el extracto de código de la *figura 25,* perteneciente a la cinemática directa, se utiliza el teorema del coseno para calcular los puntos finales. “Alfa” es el ángulo del servomotor principal y “betaprima” el del secundario. “DtR” es un valor constante utilizado para pasar de grados a radianes, calculado con la división del valor pi entre 180, para poder resolver las ecuaciones dado que el programa las realiza en radianes.

```
float alfa[4];
float beta[4];
float betaprima[4];
const float l1=11.5;
const float l2=12.5;
const float DtR=0.01745329;

alfa[i]=ang[i*2]*DtR;
beta[i]=ang[i*2+1]*DtR;
betaprima[i]=beta[i]-(90*DtR);
```

Figura 25.Calculo de punto final en cinemática directa.

La nomenclatura utilizada para asignar el valor de los ángulos en el extracto de código de la *figura 26,* perteneciente a la cinemática directa es la siguiente. Dado que los valores de los ocho ángulos están recogidos en un vector llamado “ang”, cada uno de ellos hay que asociárselo al correspondiente servomotor. En la primera línea se le asignan los valores del vector “ang”, en las posiciones pares de “l”, a los servomotores principales y las impares a los servomotores secundarios.

```
servoMotor[k].write(ang[l]*fit);
k++;
servoMotor[k].write(180-(ang[l+1]*fit));
```

Figura 26.Asignación de ángulos en cinemática directa.

7. Desarrollo del prototipo, imprevistos y soluciones adoptadas.

El primer inconveniente que se nos plantea es la limitación económica, dado que hubiera sido mucho más sencillo comprar los manipuladores ya hechos y ahorrarnos el proceso de fabricación de estos. Los manipuladores disponibles no se acatan a nuestras necesidades dado que la mayoría viene con funciones que no necesitamos como pueden ser manipuladores con controladores o con más grados de libertad, por lo que no es óptimo para nuestro proyecto y, además, son mucho más caros que lo invertido para la fabricación de los manipuladores en función a nuestras necesidades.

Al comprar los servomotores nos dimos cuenta de que sólo giran 180^{O}, lo cual supone un problema ya que limita mucho la posición final, dado que el efector final no puede posicionarse en distancias más próximas a 11,5 cm. del servomotor principal. Por otro lado, a nivel práctico, gira más de 180^{O} de manera imprecisa (la precisión se corrige en el código) llegando a un máximo real de 210^{O}.

La silicona no funcionó para pegar las placas a los servomotores, lo cual nos complicó la fabricación del manipulador al vernos obligados a perforar las placas y atornillarlas con arandelas y tuercas a los servomotores.

Debido al propio peso de las placas y los servomotores, se crea un pandeo en las mismas que hace que la placa secundaria toque con la plataforma, lo cual corregimos utilizando los tacos de elevación. Aunque esto no fue suficiente y recurrimos a la utilización de placas de aluminio en la placa principal.

En cuanto a los tacos elevadores, eran demasiado bajos, ya que la placa secundaria de los manipuladores hacía contacto con la plataforma y, además, uno de ellos cedió y se partió con el paso del tiempo. La solución fue cambiar los tacos por otros nuevos, los cuales tuvimos que fabricar nuevamente.

Otro problema que encontramos tras la fabricación y montaje fue que se hizo una medición errónea (1 cm. menos) de la placa de aluminio y esto genera el problema de no cumplir con uno de los requisitos, y es que no supera el punto central de la plataforma, en lugar de ello su punto final tiene como máximo el centro de esta.

Cuando empezamos a programar, la única manera para cumplir con el requisito de leer las entradas desde un fichero ".txt" era mediante el uso de un módulo adaptador de tarjeta micro-SD (o SD) para Arduino, el cual tuvimos que comprar y programar su lectura.

Debido a la baja intensidad que proporciona la placa Arduino, tuvimos que utilizar una fuente de alimentación externa, lo cual tampoco tenía efecto a la hora de mover todos los servomotores a la vez. Por esto y porque a la hora de compilar los códigos de prueba, las placas de los manipuladores colisionaban entre sí, decidimos conectar dos servomotores a la placa Arduino y los otros seis a la fuente de alimentación, de manera que probábamos los códigos con un solo manipulador mientras la fuente estaba apagada. De esta manera solucionamos los dos inconvenientes, ya que al usar solo un manipulador no chocaba con los otros y además, al dividir la intensidad entre las dos fuentes, podíamos maniobrar con todos los servomotores simultáneamente.

8. Conclusiones.

Como en este proyecto empezamos desde cero, tuvimos que fabricar la plataforma, los tacos y las placas, lo cual nos llevó bastante tiempo. Por otro lado, también tuvimos que pedir los servomotores a una tienda de China y tardaron cerca de dos meses en llegar, lo que supuso un gran retraso a la hora de empezar a realizar el proyecto.

Una vez conseguido todos los componentes, montamos la plataforma con los servomotores a la vez que aprendíamos a programar diversas funciones en Arduino. Cuando acabamos todo el montaje nos dedicamos exclusivamente a la programación de los servomotores para cumplir los diversos objetivos.

Finalmente logramos terminar la fabricación y codificación de la plataforma robótica con los cuatro manipuladores (de 2 grados de libertad cada uno). Logrando cumplimentar así con dos de los tres objetivos propuestos inicialmente, dado que los servomotores que adquirimos no pueden variar su velocidad si no es regulando el potenciómetro que se encuentra en el interior de estos. Cabe destacar que a la hora de fabricar las placas, hubo un error de medición y tampoco cumplimos con la condición de que la longitud del manipulador fuera mayor que el radio de la plataforma.

No obstante, al ver que no podíamos solucionar lo anterior y en vistas de que cumplimos los objetivos con tiempo de sobra, decidimos mejorar y clarificar los códigos en la medida de lo posible, por ejemplo, añadiendo la configuración codo arriba a los manipuladores, obteniendo así un mayor rango de alcance en el efector final.

En cuanto al tiempo que nos llevó acabarlo, estimamos unas 140 horas con la fabricación y montaje y otras 40 horas con la redacción de la memoria.

9. Conclusions.

From the beginning, the idea of this project was to develop a prototype with the robotic manipulators. So that, we had to make the platform, the supports and the links, which took a lot of time. On the other hand, we also had to request the servomotors to a Chinese store and they took about two months to arrive, which was a great delay for starting the project.

Once we got all the components, we set up the platform with the servomotors, at the same time as we learned to program various functions with Arduino. When we finished the assembly, we dedicatedour time exclusively to programming the servomotors forachieving the diverse objectives.

Finally, we finished the manufacture and codification of the robotic platform with the four manipulators (of 2 DOF each). But we just could get two of the three objectives initially proposed, because the servomotors we bought cannot change their speed unlesswe regulated the potentiometer inside them. It is important to say that when we made the plates, there was an error with the measure, and we neither get the condition that the length of the manipulators were longer than the radius of the platform.

However, seeing that we could not solve it and we achieved the other objectives with time to spare, we improved and clarified the codes as much as possible. For example, we added the “elbow up” configuration to the manipulators, then we could reach more position with the final effector.

In terms of time, the project took us approximately140 hours with the manufacture and assembly and 40 hours more writing the memory.

10. Posibles mejoras.

La mejora más importante que se le puede añadir al proyecto es la implementación de unos servomotores que puedan girar 360º, ya que así solucionaría el problema de rangos inalcanzables próximos al servomotor principal.

Otra mejora del proyecto para cumplir con los requisitos propuestos sería alargar las placas para sobrepasar el centro y cumplir así dicho requisito. Otra posible solución es reducir la superficie de la plataforma.

En cuanto a la sujeción de las placas, los tornillos no logran fijarla al 100%, por lo que una mejora importante es la total fijación de las placas a los servomotores. Esto se puede conseguir mediante la unificación de la placa con el acomodador, buscando o diseñando un acomodador equivalente de metal para poder soldarlo a la placa, las cuales han de ser todas de aluminio.

11. Bibliografía.

- “Cinemática y dinámica de robots manipuladores”, Autor: Miranda Colorado, Roger.Ed.: Marcombo, S.A. 1a Edición.
- http://wiki.robotica.webs.upv.es/wiki-de-robotica/cinematica/
- http://www.universoformulas.com/matematicas/trigonometria/teorema-coseno/
- https://forum.arduino.cc/
- https://www.luisllamas.es/tarjeta-micro-sd-arduino/
- http://www.ebay.es/itm/MG996R-MG996-Metal-Gear-RC-Servo-High-Speed-Torque-RC-CAR-1-8-for-Arduino-/222153978226?hash=item33b968b172:g:J8YAAOSw6hNZfhdm
- http://www.ebay.es/itm/Multifunction-Servo-Bracket-PTZ-Robotic-Manipulator-DIY-Robot-Mount-/272565929640?hash=item3f763206a8:g:W3kAAOSwTuJYrjBJ

11.1. Aplicaciones y programas utilizados.

- Arduino. Fue el programa que más se utilizó, ya que todo el código y la programación se hicieron mediante dicho programa.
- Word. Usado para la redacción de la memoria del proyecto.
- Sketch Up. Se utilizó sólo para hacer la representación esquemática del proyecto.
- Fritzing. Utilizado para dibujar la presentación de las conexiones del proyecto.
- Paint. Este programa se utilizó para dibujar algunos esquemas explicativos en la memoria.

12. Presupuesto.

El presupuesto a calcular de la plataforma robótica consistirá en la suma del precio de los materiales para su fabricación más el precio de la mano de obra correspondiente.

Presupuesto.			
	Precio	Nº de unidades	Total.
Servomotor.	7,19 €	8	57,52 €
Carcasa.	5,91 €	8	47,28 €
Placa Arduino.**	10,00 €	1	10,00 €
Adaptador de micro-SD.	5,00 €	1	5,00 €
Plataforma, tacos y patas de sujeción. *	25,00 €	1	25,00 €
Fuente de alimentación.*	150,00 €	1	150,00 €
Mano de obra.	45€/hora	120 horas	5.400,00 €
		TOTAL:	5.694,80 €

*Precios medios o estimados.

**El precio de la placa Arduino es de 10€ incluyendo en este una protoboard y cables para las conexiones.

13. Anexos.

13.1. Anexo I. Código.

13.1.1. CinemáticaDirecta

```
//  Incluimos la librería para poder controlar el servo y la microSD.
#include <Servo.h>
#include <SD.h>
//  Creamos el fichero para utilizar el archivo txt de la microSD en el programa.
File myFile;
//  Declaramos variables globales para ultilizar tanto en el void como en el loop.
intPosicion=0;
int j=0;
float ang[7];
//  Declaramos un vector para controlar los servomotores.
Servo servoMotor[8];
voidsetup() {
//  Declaramos las variables que utilizaremos en el setup.
float x[4];
  float y[4];
  float x0[4];
float y0[4];
float R=23.9;
//  Asignamos pines a servomotores.
servoMotor[0].attach(2);
servoMotor[1].attach(3);
servoMotor[2].attach(5);
servoMotor[3].attach(6);
servoMotor[4].attach(7);
servoMotor[5].attach(8);
servoMotor[6].attach(9);
servoMotor[7].attach(10);
//  Inicializamoslosservomotoresen 0º.
for(inti=0;i<8;i++){
   if(i%2==0){
servoMotor[i].write(0);
}else
   {
servoMotor[i].write(180-0);
   }
 }
//  Iniciamos el monitor serie para mostrar el resultado
Serial.begin(9600);
Serial.print(F("Iniciando microSD ..."));
//  En caso de no poder reconocer la microSD, se muestra por el monitor.
if(!SD.begin(4)){
Serial.println(F("No se pudo inicializar"));
return;
 }
Serial.println(F("inicializacion correcta"));
//  Utilizamos el comando "remove" para eliminar el archivo y poder escribir en el los
```

```
// ángulos deseados.
SD.remove("nameFile.txt");
// En el caso de que no se disponga de archivos dentro, o queramos crear alguno nuevo
// desdeaqui. Utilizamos lo siguiente comentado.
// Asociamos la apertura (o creación en el caso de que no exista) del archivo nameFile al
// ficheromyFile.
myFile = SD.open("nameFile.txt",FILE_WRITE);
if (myFile){
// Se escribe información en el documento de texto nameFile.txt.
myFile.print("0 180 0 180 0 180 0 180\n");
// Se cierra el archivo para almacenar los datos.
myFile.close();
}else
 {
Serial.println(F("No se abrió correctamente"));
myFile.close();
  }
// Asociamos la apertura del archivo nameFile al fichero myFile.
myFile = SD.open("nameFile.txt");
// Declaramos un entero con el valor del tamaño del archivo.
inttotalBytes=myFile.size();
// Declaramos una cadena de caracteres vacía que iremos rellenando con los valores del
// archivo.
 String cadena="";
 if (myFile){
// Ahora iremos leyendo caracter por caracter del fichero. De tal modo que cuando llegue
// a la última posición (que será el tamaño total del fichero), vuelva a empezar.
if(Posicion>=totalBytes){
Posicion=0;
  }
// Nos posicionamos en "Posicion", que será 0 la primera vez y cada vez que recorra todo
// el fichero.
myFile.seek(Posicion);
// Mientras el archivo "myFile" esté disponible para leer, creamos un caracter que vaya
// leyendo individualmente todos los caracteres que hay en el archivo.
// Luego concatenamos los caracteres enteros en la cadena creada anteriormente, para
// luegoalmacenarlos en un entero después de reconocer un espacio o una coma.
// Cabe destacar que al reconocer un espacio o una coma, se borra la cadena para volver a
// estar disponible a la hora de almacenar el siguiente entero.
// Los enteros se almacenan en un vector llamado "ang[]".
while (myFile.available()){
    char caracter=myFile.read();
cadena=cadena+caracter;
if(isSpace(caracter)==1 || caracter==',')
    {
// Como la cadena es un String y el vector un entero con decimal, tenemos que hacer la
// conversión mediante el comando "toFloat".
ang[j]= cadena.toFloat();
Serial.print(F("ang"));
Serial.print(j);
Serial.print(F(" es = "));
Serial.println(ang[j]);
```

```
Serial.println(F("-----------------"));
delay(100);
j++;
//  "Reseteamos" la cadena para que quede vacía y vuelva a almacenar el siguiente entero
//  delnúmero del archivo en el vector.
      cadena="";
    }
//  Si el caracter es un punto o un salto de línea, acabamos la lectura. 10 y 46 en ASCII
//  corresponden a un salto de línea y a un punto respectivamente.
if(caracter==10 || caracter=='.' || caracter ==46)
{
      break;
    }
   }
 }
//  Calculamos los puntos finales que alcanzan los manipuladores para luego mostrarlos
//  por el monitor.
for(int i=0;i<4;i++){
//  Inicializamos las constantes y variables locales.
float alfa[4];
   float beta[4];
   float betaprima[4];
const float l1=11.5;
const float l2=12.5;
//  "DtR" "Degree to Radian" es un valor fijo resultante de la división de pi entre 180 para
//  pasarde grados a radianes.
const float DtR=0.01745329;
alfa[i]=ang[i*2]*DtR;
   beta[i]=ang[i*2+1]*DtR;
betaprima[i]=beta[i]-(90*DtR);
   x[i]=l1*cos(alfa[i])+l2*cos(alfa[i]+betaprima[i]);
y[i]=l1*sin(alfa[i])+l2*sin(alfa[i]+betaprima[i]);
Serial.print(F("la posición del manipulador '"));
Serial.print(i+1);
Serial.print(F("' con respecto a su sistema de referencia es: ("));
Serial.print(x[i]);
Serial.print(F(","));
Serial.print(y[i]);
Serial.println(F(")."));
Serial.println(F("++++++++++++++++++++++++"));
 }
//  Pasamos las coordenadas locales al sistema de referencia central.
 x0[0]=x[0];
 y0[0]=y[0]-R;
 x0[1]=R-y[1];
 y0[1]=x[1];
 x0[2]=-x[2];
 y0[2]=R-y[2];
 x0[3]=y[3]-R;
 y0[3]=-x[3];
for(int i=0;i<4;i++)
 {
```

```
    Serial.print(F("la posición del manipulador '"));
    Serial.print(i+1);
    Serial.print(F("' con respecto al sistema de referencia global es: ("));
    Serial.print(x0[i]);
    Serial.print(F(","));
    Serial.print(y0[i]);
    Serial.println(F(")."));
    Serial.println(F("--------------------------------"));
     }
    }
    void loop() {
    int k=0;
    //  Utilizamos el entero decimal constante "fit" para ajustar la imprecisión de los angulos
    //  de los servomotores.
    constfloatfit=0.86;
    //  Colocamos el ángulo leido en los enteros correspondientes del vector "ang[]" en los
    //  servomotores.
    //  Los servomotores secundarios están "boca abajo", por lo que el  ángulo es opuesto.
    //  Para que esto no afecte al usuario, nosotros lo invertimos aquí. Si los ángulos de los
    //  servomotoresestán fuera del rango (0, 210) indicamos por el monitor que se le ha dado
    //  una posición inalcanzable. En este caso el manipulador se coloca en los ángulos 0,0
    for(int l; l<8; l+=2){
       if (ang[l]<0 || ang[l]>210 || ang[l+1]<0 || ang[l+1]>210)
    {
    Serial.println(F("Posición inalcanzable. Utilice ángulos que estén dentro del rango (0,
210)"));
    servoMotor[k].write(0);
    delay(1000);
        k++;
    servoMotor[k].write(180);
    delay(1000);
        k++;
    }else{
    servoMotor[k].write(ang[l]*fit);
    Serial.print(F("Colocamos el ángulo [ "));
    Serial.print(ang[l]);
    Serial.print(F(" ]"));
    Serial.print(F("en el servomotor principal del manipulador '"));
    Serial.print(l/2+1);
    Serial.println(F("'"));
        k++;
    delay(1000);
    servoMotor[k].write(180-(ang[l+1]*fit));
    Serial.print(F("Colocamos el ángulo [ "));
    Serial.print(ang[l+1]);
    Serial.print(F(" ]"));
    Serial.print(F("en el servomotor secundario del manipulador '"));
    Serial.print(l/2+1);
    Serial.println(F("'"));
        k++;
    delay(1000);
       }
```

```
 }
delay(500000);
}
```

13.1.2. Cinemática Inversa

```
//  Incluímos la librería para poder controlar el servo y la microSD.
#include <SD.h>
#include <Servo.h>
//  Creamos el fichero para utilizar el archivo txt de la microSD en el programa.
File myFile;
//  Declaramos las variables globales para utilizar tanto en el setup como en el loop.
float a[4];
float b[4];
inti=0;
int k=0;
//  Declaramos un vector para controlar los servomotores.
Servo servoMotor[8];
void setup() {
//  Asignamos pines a servomotores.
servoMotor[0].attach(2);
servoMotor[1].attach(3);
servoMotor[2].attach(5);
servoMotor[3].attach(6);
servoMotor[4].attach(7);
servoMotor[5].attach(8);
servoMotor[6].attach(9);
servoMotor[7].attach(10);
//  Inicializamos los servomotores en 0º.
for(inti=0;i<8;i++){
   if(i%2==0){
servoMotor[i].write(0);
}else
   {
servoMotor[i].write(180-0);
}
 }
//  Declaramos variables locales.
floatvar[8];
float x[4];
float y[4];
intPosicion=0;
int j=0;
//  Declaramos las constantes.
//  "RtD" "Radian to Degree" es un valor fijo resultante de la división de 180 entre pi para
// pasar de radianes a grados utilizado posteriormente .
const float l1=11.5;
const float l2=12.5;
const float R=23.9;
const float RtD=57.29577951;

//  Iniciamos el monitor serie para mostrar el resultado
Serial.begin(9600);
```

```
Serial.print(F("Iniciando microSD ..."));
// En caso de no poder reconocer la microSD, nos lo muestra por el monitor.
if(!SD.begin(4)) {
Serial.println(F("No se pudo inicializar"));
return;
 }
Serial.println(F("inicializacion correcta"));
// Utilizamos el comando "remove" para eliminar el archivo y poder escribir en el los
// ángulosdeseados.
SD.remove("nameFile.txt");
// En el caso de que no se disponga de archivos dentro, o queramos crear alguno nuevo
// desdeaqui. Utilizamos lo siguiente comentado.
// Asociamos la apertura (o creación en el caso de que no exista) del archivo nameFile al
// ficheromyFile.
myFile = SD.open("nameFile.txt",FILE_WRITE);
if (myFile){
// Se escribe información en el documento de texto datos.txt.
myFile.print("(10,-15) (3,4) (-3,4) (-3,-3) .");
// Se cierra el archivo para almacenar los datos.
myFile.close();
}else
 {
Serial.println(F("No se abrió correctamente"));
myFile.close();
 }
// Asociamos la apertura del archivo nameFile al fichero myFile.
myFile = SD.open("nameFile.txt");
// Declaramos un entero con el valor del tamaño del archivo.
inttotalBytes=myFile.size();
// Declaramos una cadena de caracteres vacía que iremos rellenando con los valores del
// archivo.
 String cadena="";
 if(myFile) {
// Ahora iremos leyendo caracter por caracter del fichero. De tal modo que cuando llegue
// a la última posición (que será el tamaño total del fichero), vuelva a empezar.
if(Posicion>=totalBytes) {
Posicion=0;
    }
// Nos posicionamos en "Posicion", que será 0 la primera vez y cada vez que recorra todo
// el fichero.
myFile.seek(Posicion);
// Mientras el archivo "myFile" esté disponible para leer, creamos un caracter que vaya
// leyendo individualmente todos los caracteres que hay en el archivo.
// Luego concatenamos los caracteres enteros en la cadena creada anteriormente, para
// luegoalmacenarlos en un entero después de reconocer un espacio o una coma.
// Cabe destacar que al reconocer un espacio o una coma, se borra la cadena para volver a
// estar disponible a la hora de almacenar el siguiente entero.
// Los enteros se almacenan en un vector llamado "var[]".
while (myFile.available()) {
    char caracter=myFile.read();
cadena=cadena+caracter;
// Para evitar errores de lectura, borramos el paréntesis abierto cada vez que se detecte,
```

```
//  que es el que nos da problemas de operaciones.
if(caracter=='(')
    {
cadena="";
    }
    if(isSpace(caracter)==1 || caracter==','){
//  Como la cadena es un String y el vector un entero con decimal, tenemos que hacer la
//  conversión mediante el comando "toFloat".
var[j]= cadena.toFloat();
Serial.print(F("var"));
Serial.print(j);
Serial.print(F(" es = "));
Serial.println(var[j]);
Serial.println(F("-----------------"));
delay(100);
j++;
//  "Reseteamos" la cadena para que quede vacía y vuelva a almacenar el siguiente entero
//  delnúmero del archivo en el vector.
      cadena="";
    }
//  Si el caracter es un punto o un salto de línea, acabamos la lectura. 10 y 46 en ASCII
//  corresponden a un salto de línea y a un punto respectivamente.
if(caracter==10 || caracter=='.' || caracter ==46)
{
      break;
    }
   }
 }
//  Ahora asignamos las variables leídas a valores de coordenas y a su vez, aplicamos los
//  cambios de sistemas de referencia corresponientes de tal modo que solo haya un único
//  sistema de referencia en el centro de la plataforma y las coordenadas sólo se basen en
//  unúnico sistema.
  x[0]=var[0];
Serial.print(F("x[0]= "));
delay(50);
Serial.println(x[0]);
delay(50);
y[0]=R+var[1];
Serial.print(F("y[0]= "));
delay(50);
Serial.println(y[0]);
delay(50);
  x[1]=var[3];
Serial.print(F("x[1]= "));
delay(50);
Serial.println(x[1]);
delay(50);
  y[1]=R-var[2];
Serial.print(F("y[1]= "));
delay(50);
Serial.println(y[1]);
delay(50);
```

```
  x[2]=-var[4];
Serial.print(F("x[2]= "));
delay(50);
Serial.println(x[2]);
delay(50);
  y[2]=R-var[5];
Serial.print(F("y[2]= "));
delay(50);
Serial.println(y[2]);
delay(50);
  x[3]=-var[7];
Serial.print(F("x[3]= "));
delay(50);
Serial.println(x[3]);
delay(50);
  y[3]=R+var[6];
Serial.print(F("y[3]= "));
delay(50);
Serial.println(y[3]);
delay(50);
Serial.println(F("-----------------"));
for(inti=0; i<4; i++){
//  Utilizamos las siguientes ecuaciones para obtener los ángulos a asignar a los
//  servomotores.
   a[i]=((acos(x[i]/sqrt(x[i]*x[i]+y[i]*y[i])))+acos((-
l1*l1+l2*l2+x[i]*x[i]+y[i]*y[i])/(2*l2*sqrt(x[i]*x[i]+y[i]*y[i]))))*RtD;
   b[i]=((acos((l1*l1+l2*l2-x[i]*x[i]-y[i]*y[i])/(2*l1*l2)))*RtD)-90;
   if (a[i]<0 || a[i]>210 || b[i]<0 || b[i]>210){
if (x[i]<=0){
Serial.println(F("El codo abajo no es válido"));
Serial.println(F("---------Para este manipulador utilizamos codo ARRIBA----------"));
Serial.print(F("a["));
Serial.print(i);
Serial.print(F("]="));
Serial.println(a[i]);
delay(50);
Serial.print(F("b["));
Serial.print(i);
Serial.print(F("]="));
Serial.println(b[i]);
delay(50);
}else{
Serial.println(F("el codo arriba no es válido"));
Serial.println(F("---------Para este manipulador utilizamos codo ABAJO----------"));
Serial.print(F("a["));
Serial.print(i);
Serial.print(F("]="));
Serial.println(a[i]);
delay(50);
Serial.print(F("b["));
Serial.print(i);
Serial.print(F("]="));
```

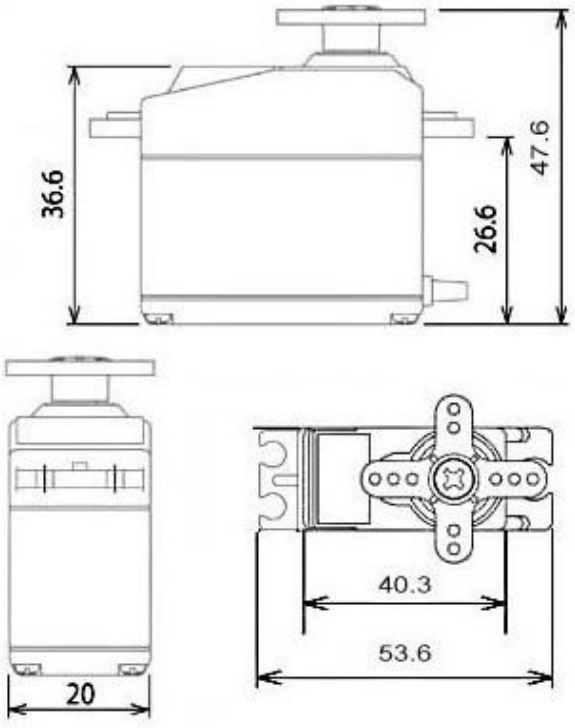
```
    Serial.println(b[i]);
    delay(50);
    }
    //  Utilizamos las siguientes ecuaciones para obtener los ángulos a asignar a los
    //  servomotores.
        a[i]=(((acos(x[i]/sqrt(x[i]*x[i]+y[i]*y[i])))-acos((-
l1*l1+l2*l2+x[i]*x[i]+y[i]*y[i])/(2*l2*sqrt(x[i]*x[i]+y[i]*y[i]))))*RtD);
        b[i]=270-((acos((l1*l1+l2*l2-x[i]*x[i]-y[i]*y[i])/(2*l1*l2)))*RtD);
    }else
       {
    if(x[i]<=0){
    Serial.println(F("---------Para este manipulador utilizamos codo ABAJO----------"));
    Serial.print(F("a["));
    Serial.print(i);
    Serial.print(F("]="));
    Serial.println(a[i]);
    delay(50);
    Serial.print(F("b["));
    Serial.print(i);
    Serial.print(F("]="));
    Serial.println(b[i]);
    delay(50);
    }else{
    Serial.println(F("---------Para este manipulador utilizamos codo ARRIBA----------"));
    Serial.print(F("a["));
    Serial.print(i);
    Serial.print(F("]="));
    Serial.println(a[i]);
    delay(50);
    Serial.print(F("b["));
    Serial.print(i);
    Serial.print(F("]="));
    Serial.println(b[i]);
    delay(50);
    }
       }
     }
    }
    voidloop() {
    //  Utilizamos el entero decimal constante "fit" para ajustar la imprecisión de los angulos
    //  de los servomotores.
    constfloatfit=0.86;
    //  Colocamos los ángulos calculados anteriormente en los servomotores.
    //  Los servomotores secundarios están "boca abajo", por lo que el ángulo es opuesto. Para
    //  queesto no afecte al usuario, se invierte en este último paso. Si los ángulos de los
    //  servomotoresestán fuera del rango (0, 210) indicamos por el monitor que se le ha dado
    //  una posición inalcanzable. En este caso el manipulador se coloca en los ángulos 0,0
    for(i=0;i<4;i++){
    if (a[i]<0 || a[i]>210 || b[i]<0 || b[i]>210)
    {
    Serial.print(F("La posición indicada es inalcanzable. Algún ángulo de los servomotores del
manipulador '"));
```

```
Serial.print(i+1);
Serial.println(F("' es inferior a 0 o superior a 210."));
servoMotor[k].write(0);
delay(500);
    k++;
servoMotor[k].write(180);
delay(500);
    k++;
}else{
Serial.println(F("-----------------"));
Serial.print(F("a["));
Serial.print(i);
Serial.print(F("] es "));
delay(100);
Serial.println(a[i]);
delay(500);
servoMotor[k].write(a[i]*fit);
    k++;
Serial.println(F("-----------------"));
Serial.print(F("b["));
Serial.print(i);
Serial.print(F("] es "));
delay(500);
Serial.println(b[i]);
delay(500);
servoMotor[k].write(180-(b[i]*fit));
    k++;
   }
 }
delay(500000);
}
```

13.2. Anexo II. Datasheet Servomotor.

MG996R High Torque

Metal Gear Dual Ball Bearing Servo



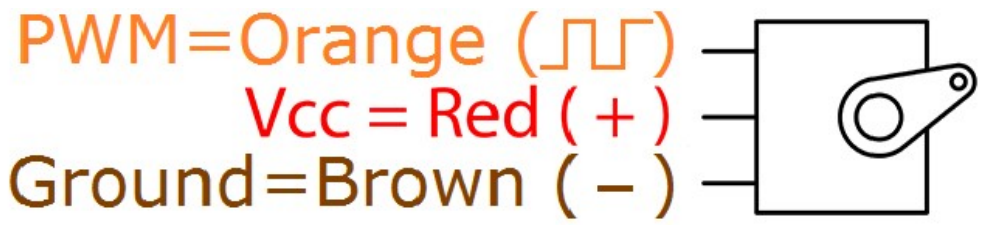
This High-Torque MG996R Digital Servo features metal gearing resulting in extra high 10kg stalling torque in a tiny package. The MG996R is essentially an upgraded version of the famous MG995 servo, and features upgraded shock-proofing and a redesigned PCB and IC control system that make it much more accurate than its predecessor. The gearing and motor have also been upgraded to improve dead bandwith and centering. The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.

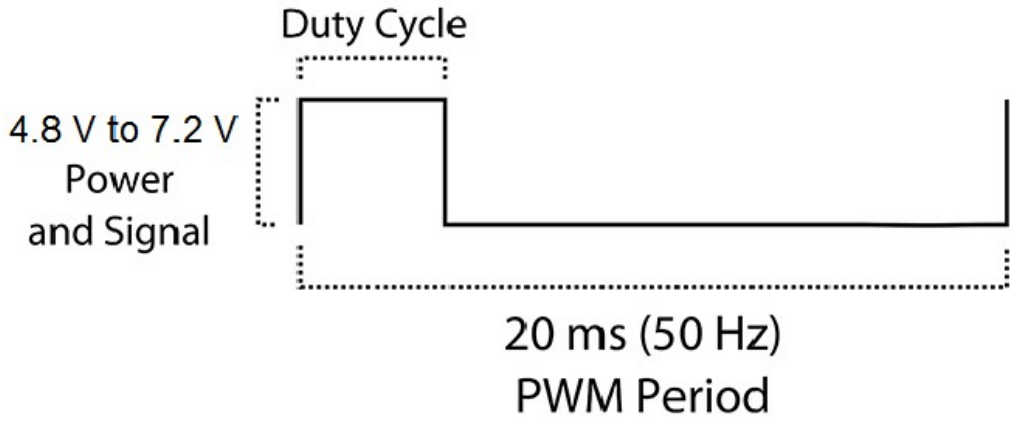
This high-torque standard servo can rotate approximately 120 degrees (60 in each direction). You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG996R Metal Gear

Servo also comes with a selection of arms and hardware to get you set up nice and fast!

Specifications

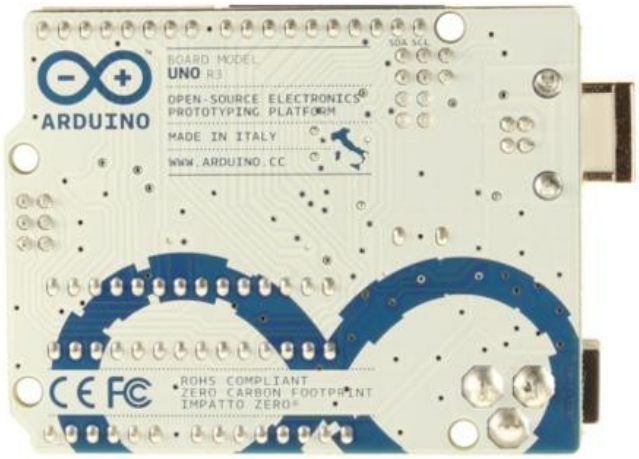
- Weight: 55g
- Dimension: 40.7 x 19.7 x 42.9 mmapprox.
- Stall torque: 9.4 kgf·cm (4.8 V), 11 kgf·cm (6V)
- Operating speed: 0.17 s/60º (4.8 V), 0.14 s/60º (6V)
- Operating voltage: 4.8 V a 7.2V
- Running Current 500 mA – 900 mA (6V)
- StallCurrent 2.5 A(6V)
- Dead band width: 5µs
- Stable and shock proof double ball bearingdesign
- Temperaturerange: 0 ºC – 55ºC





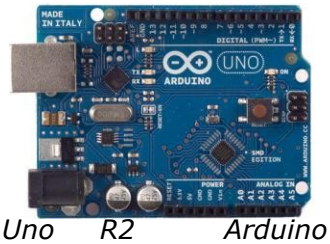
13.3. Anexo III. Placa Arduino UNO.

Arduino Uno R3 Front



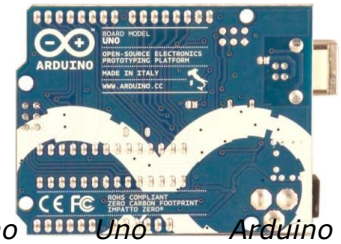
Arduino Unc R3Back

Arduino Uno R2 Front



Arduino Uno SMD

Arduino Uno Front



Arduino Uno Back

Overview

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

Revision 2of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode.

Revision 3of the board has the following new features:

- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltageprovided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for futurepurposes.
- StrongerRESETcircuit.
- Atmega 16U2 replacethe8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Schematic& Reference Design

EAGLE files: arduino-uno-Rev3-reference-design.zip(NOTE: works with Eagle 6.0 and newer) Schematic: arduino-uno-Rev3-schematic.pdf

Note: The Arduino reference design can use an Atmega8, 168, or 328, Current models use an ATmega328, but an Atmega8 is shown in the schematic for reference. The pin configuration is identical on all three processors.

Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through thispin.
- **5V.**This pin outputs a regulated 5V from the regulator on the board. The board can besupplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. Wedon'tadviseit.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50mA.
- **GND.** Groundpins.

Memory

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead()functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40

mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serialchip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. Seetheattachinterrupt()functionfordetails.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with theanalogWrite()function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using theSPIlibrary.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it'soff.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- **TWI: A4 or SDA pin and A5 or SCL pin.** Support TWI communication using theWirelibrary.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Usedwithanalogreference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on theboard.

See also the mapping between Arduino pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows,a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the

Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSeriallibraryallows for serial communication on any of the Uno's digital pins.

The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentationfor details. For SPI communication, use the SPI library.

Programming

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the referenceandtutorials.

The ATmega328 on the Arduino Uno comes preburned with a bootloaderthat allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructionsfor details.

The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line toground, making it easier to put into DFUmode.

You can then use Atmel's FLIP software(Windows) or the DFU programmer(Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorialfor more information.

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half- second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending thisdata.

The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum threadfor details.

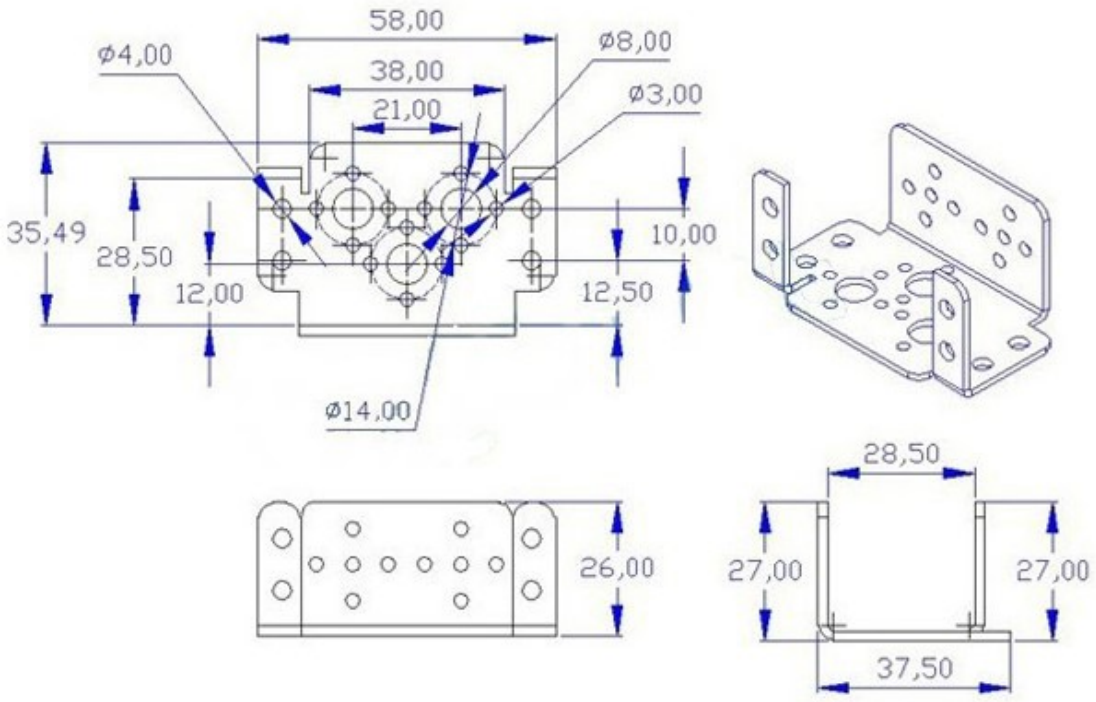
USB Overcurrent Protection

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics

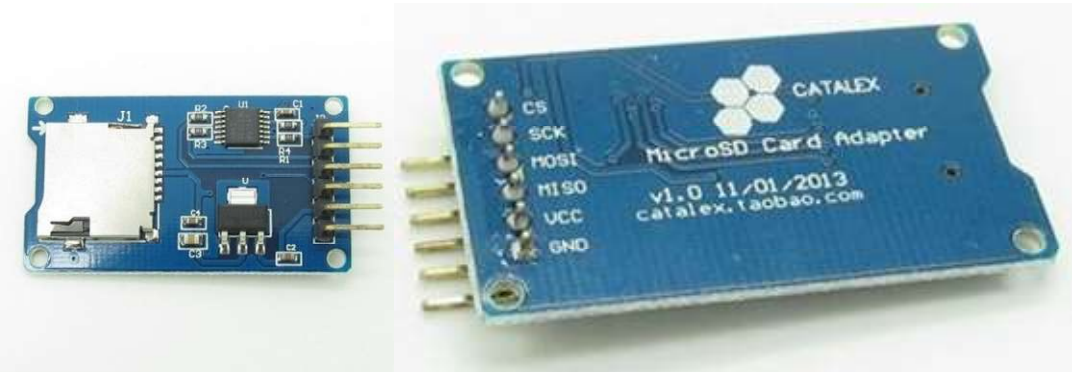
The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the boardto be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the otherpins.

13.4. Anexo IV. Especificaciones de la carcasa exterior.



13.5. Anexo V. Módulo adaptador de micro-SD.

Micro SD Card Micro SDHC Mini TF Card Adapter Reader Module for Arduino



Description

- The module (MicroSD Card Adapter) is a Micro SD card reader module for reading and writing through the file system and the SPI interface driver, SCM system can be completed within a file MicroSDcard
- Support Micro SD Card, Micro SDHC card (high speedcard)
- Level conversion circuit board that can interface level is 5V or3.3V
- Power supply is 4.5V ~ 5.5V, 3.3V voltage regulator circuitboard
- Communications interface is a standard SPI interface
- 4 M2 screws positioning holes for easyinstallation
- Control Interface: A total of six pins (GND, VCC, MISO, MOSI, SCK, CS),GND to ground, VCC is the power supply, MISO, MOSI, SCK for SPI bus, CS is the chip select signalpin;
- 3.3V regulator circuit: LDO regulator output 3.3V for level conversion chip,Micro SD cardsupply;
- Level conversion circuit: Micro SD card to signal the direction of converts 3.3V, MicroSD card interface to control the direction of the MISO signal is also converted to 3.3V, general AVR microcontroller systems can read thesignal;
- Micro SD card connector: self bomb deck, easy cardinsertion.
- Positioning holes: 4 M2 screws positioning holes with a diameter of 2.2mm,so the module is easy to install positioning, to achieve inter-modulecombination.

Interface Parameters:

Items	Min	Typical	Max	Unit
Power Voltage VCC	4.5	**5**	5.5	V
Current	0.2	**80**	200	mA
Interface Electrical Potential	3.3 or 5			V
Support Card Type	Micro SD Card(<=2G), Mirco SDHC Card(<=32G)			—
Size	42X24X12			mm
Weight	5			g

Micro SD Card Interface Module: