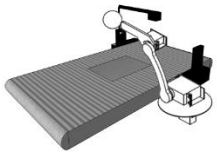
Banda de transporte

La estación se transporte será el inicio de la celda de manufactura. Consiste en una banda transportadora que transporta objetos de diferente tonalidad (blanco y negro) y un sensor de proximidad que es activado cuando el objeto llega al límite de la banda. Además contará con un actuador junto con su driver y para todo el sistema se utilizará un microcontrolador.

Objetivo

Iniciar movimiento de banda mediante actuador (motor DC), el sensor de proximidad detecta objeto y comienza un tiempo de detención del movimiento al mismo momento en que el objeto se transporta a su posición final en la banda. Al final, cuando el tiempo se cumple, la banda se detiene y se manda una señal a la siguiente estación para comenzar. Para inicializarla se realiza mediante un protocolo de comunicación entre la banda de transporte y el tablero de control, al igual que para la parada normal. Para la parada de emergencia, se activa en el tablero de control y se detiene la banda pero mediante una interrupción.



Componentes

- 1 microcontrolador Arduino UNO (ATmega328P)
- Motor DC de alto torque, 38 RMP, 3.6 kg/cm (B01-1:220. MOTORREDUCTOR)
- Driver para motor DC (L298N)
- Sensor infrarrojo de proximidad SHARP GP2Y0A21YK
- Material para banda
- Rodillo o tubo
- Soportes y estructura de la banda

Es necesario saber las características de operación de los componentes como el voltaje, la corriente y la potencia para poder hacer un diseño óptimo y funcional. En la tabla 1 se presentan dichas características de los componentes que se utilizarán.

Tabla 1. Características y consumo de operación de los componentes

Componente	Voltaje (V)	Corriente (mA)	Potencia (mW)*
Motorreductor de alto torque B01:220	5	75 – 670	3350
Controlador para motor DC L298N	5 – 35	30 – 4000	20000
MCU ATmega328P (Arduino UNO)	1.8 – 5.5	200	1100
Sensor de proximidad SHARP GP2Y0A21YK	5	30	150

*Potencia máxima

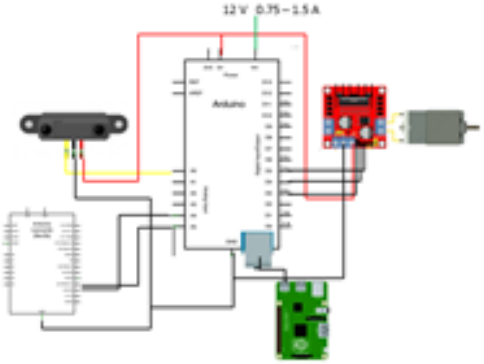
Entradas y salidas del microcontrolador:

Entradas:
Pin A0 (salida del sensor proximidad; análogo)
Pin 3 (enable del controlador de motor DC)
Pin 4 (in1 del controlador del motor DC)
Pin 5 (in2 del controlador del motor DC)

Salidas:

Pin A4 del arduino uno a Pin 3 (SCL) del arduino leonardo o a pin de SCL (protocolo i2c)
Pin A5 del arduino uno a Pin 2 (SDA) del arduino leonardo o a pin de SDA (protocolo i2c)

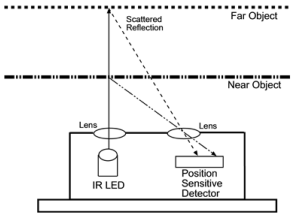
Diagrama de conexiones general



Sensor

El sensor que se utilizará en la banda transportadora es el SHARP GP2Y0A02YK0F. Es un sensor óptico de distancia compuesto por un IR LED (LED Infrarrojo) con un dispositivo de detector de posición. El sensor es analógico por lo que la salida será mediante voltaje y se tendrá que hacer una conversión para obtener el valor en distancia.

El funcionamiento consiste en que el IR LED manda una señal, es reflejada con un objeto y el dispositivo de detector de posición la recibe. Se registra el tiempo que tomó en ir y regresar la señal y en base a eso se manda la salida de tensión analógica.



La tensión de alimentación del sensor es de 4.5 a 5.5V y el consumo de corriente de 33mA. El intervalo de refresco entre mediciones es de unos 80ms. El principal obstáculo del sensor es la luminosidad del ambiente. La salida analógica tiene un valor de 2.5V a 20 cm, y de 0.4 a 150cm. Sin embargo, como hemos mencionado, la respuesta es no lineal por lo que es necesario interpolar el valor para obtener un nivel de precisión adecuado. El rango de medición va de 20 a 150 cm. Para la aplicación deseada no se medirá distancia, solamente con que detecte algún objeto. Si se desea saber un valor menor a 20 se tendrán dificultes porque se registrará 1 misma salida pero con distancias diferentes.

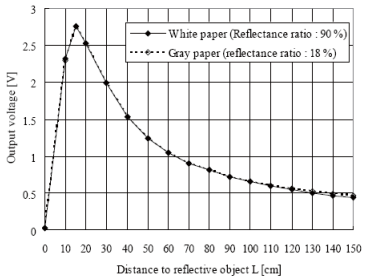
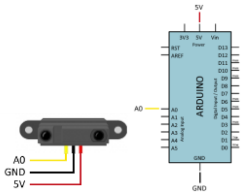


Diagrama de conexiones o terminales del sensor y las conexiones al microcontrolador.



Otras aplicaciones del sensor puede ser en sensores de posición, fin de carrera o medir distancias.

Código ejemplo

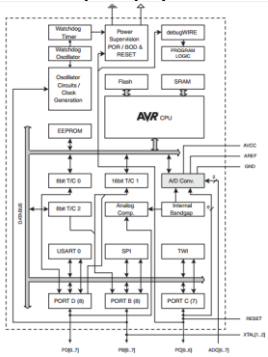
```
int d = 0;
void setup() {
  pinMode(2, OUTPUT);
  Serial.begin(9600);
}
void loop() {
  int s = analogRead(A0);
  digitalWrite(2, LOW);
  delay(500);
  d = pow(3027.4 / s, 1.2134);
  if(d >= 0 && d < 10)   {
    Serial.println(d);
    digitalWrite(2, HIGH);
   }
}
```

Microcontrolador

El microcontrolador que se va a utilizar es el Arduino UNO (ATmega328P) debido a su sencilla funcionalidad y cumple con las características para implementar la estación de transporte.

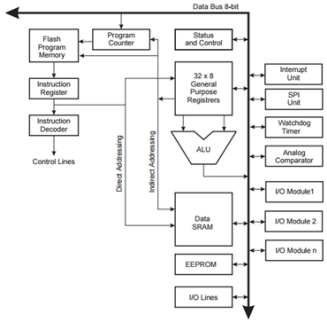
Un resumen de las características del dispositivo en el Arduino Uno (ATmega328P) son: AVR 8-bit MCU, RISC (Reduced Instruction Set Computing), 32K bytes de In-System

Programmable Flash de los cuales 0.5 KB son usados para bootloader, 1 KB EEPROM, 2 KB de RAM, una velocidad de reloj de 16 MHz, 14 pines para entradas y salidas digitales, de los cuales 6 pueden usarse como salida PWM, 6 entradas análogas y 8 canales ADC. Incluye un USART, Interfaz Serial 2-wire (I2C), puerto serial SPI.



El tipo de microprocesador AVR utiliza la arquitectura Harvard, con buses separados para la memoria de datos y la memoria de programa.

Figure 7-1. Block Diagram of the AVR Architecture



Protocolos de comunicación

Los microcontroladores empleados cuentan con pines y protocolos configurados que harán más sencilla la comunicación entre ellos. Para la banda de transporte se utilizarán 2 protocolos de comunicación: I2C (Circuito Inter-Integrado; protocolo síncrono) que será entre la banda de transporte y el manipulador robótico, y el USB (Bus Universal en Serie; protocolo síncrono) que será entre la banda de transporte y el tablero de control; en este caso solamente será unidireccional por lo que la banda de transporte sólo recibe datos del tablero de control.

<u>Códigos ejemplo</u>

Prueba comunicación I2C (Wire)

```
// MASTER READER
#include <Wire.h>
void setup()
{
  Wire.begin();
  Serial.begin(9600);
}
void loop()
{
  Wire.requestFrom(8, 6);
  while (Wire.available))
  {
    char c = Wire.read();
    Serial.print(c);
  }
  delay(500);
}

// MASTER WRITER
#include <Wire.h>
void setup()
{
  Wire.begin();
}
```

```
byte x = 0;
void loop()
{
  Wire.beginTransmission(8);
  Wire.write("x is ");
  Wire.write(x);
  Wire.endTransmission();
  x++;
  delay(500);
}

// SLAVE RECEIVER
#include <Wire.h>
void setup()
{
  Wire.begin(8);
  Wire.onReceive(receiveEvent);
  Serial.begin(9600);
}
void loop()
{
  delay(100);
}
void receiveEvent(int howMany)
{
  while (1 < Wire.available())
  {
    char c = Wire.read();
    Serial.print(c);
  }
  int x = Wire.read();
  Serial.println(x);
}

// SLAVE SENDER
#include <Wire.h>
void setup()
{
  Wire.begin(8);
  Wire.onRequest(requestEvent);
}
void loop()
{
  delay(100);
}
void requestEvent()
{
  Wire.write("hello ");
}
```

Código ejemplo

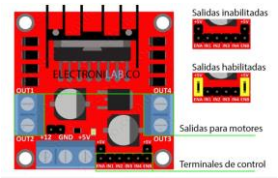
Prueba comunicación USB (Serial)

```
int incomingByte = 0;
int LED1 = 5;
int LED2 = 6;
void setup() {
  Serial.begin(9600);
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
}
void loop() {
  if (Serial.available() > 0) {
    incomingByte = Serial.read();
    if (incomingByte == 1) {
      digitalWrite(LED1, HIGH);
      delay(1000);
      digitalWrite(LED1, LOW);
      Serial.print("I received: ");
      Serial.println(incomingByte, DEC);
      }
      if else (incomingByte == 2) {
        digitalWrite(LED2, HIGH);
        delay(1000);
        digitalWrite(LED2, LOW);
        Serial.print("I received: ");
```

```
      Serial.println(incomingByte, DEC);
    }
  }
}
```

Actuador

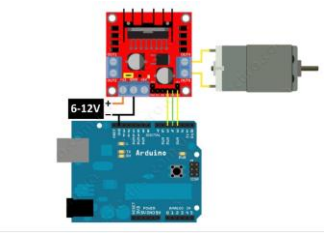
El actuador que se utilizará en el sistema es un motorreductor DC de alto torque, 38 RMP, 3.6 kg/cm, de 5 V y con un máximo de 670 mA. Para poder controlar la velocidad del motor, se utilizará un driver o controlador, el L298N. El driver dual para motores (full-bridge) L298N permite controlar dos motores de corriente continua o un motor de paso bipolar de hasta 2 amperes. El módulo cuenta con componentes de protección, un regulador LM7805 para suministrar a la parte lógica 5 V. Cuenta con conectores para habilitar las salidas del módulo. Internamente el circuito trae 4 puentes h medios (con 2 se controla un motor) y se pueden conectar motores de hasta 35 V. Un aspecto importante del controlador es que se puede controlar la dirección y la velocidad usando el PWM. Con el driver se puede usar hasta 2 motores de hasta 2 amperes por cada motor; si es un motor de menos de 4 pero más de 2 se pueden usar los puentes h conectados en paralelo 2 conectores para motor A, 2 conectores para motor B,
Los pines de control para los motores son: enable A y enable B que se habilitan para controlar los motores y controlar la velocidad de giro, y los 4 pines de control de los puentes H (IN1 (polarización directa) e IN2 (polarización inversa) para controlar el motor A, IN3 (polarización directa) e IN4 (polarización inversa) para controlar el motor B) para controlar el sentido de giro. Estas salidas llegan a los bornes de los motores.



El controlador se puede alimentar de 2 maneras. Cuando el jumper de selección de 5V se encuentra activo, la entrada de alimentación puede ser de 6 a 12 V y la salida será de 5 V, ambos DC. El voltaje puede usar en cualquier dispositivo de 5V solamente con precaución de corriente. Cuando el jumper de selección de 5V se encuentra inactivo, la alimentación del controlador varía entre 12 y 35 V DC porque el regulador no se encuentra funcionando. Debido a esto, para alimentar la parte lógica del controlador, se debe de alimentar con 5 VDC el otro conector que usualmente suele ser la misma que la parte de control.

Para controlar la velocidad del motor, tenemos que hacer uso de PWM. Este PWM será aplicado a los pines de activación de cada salida o pines ENA y ENB respectivamente, por tanto los jumper de selección no serán usados.

<u>Esquema de conexión</u>



Código ejemplo

```
int IN3 = 5;
int IN4 = 4;
int ENB = 3;
void setup()
{
 pinMode (ENB, OUTPUT);
 pinMode (IN3, OUTPUT);
 pinMode (IN4, OUTPUT);
}
void loop()
{
  digitalWrite (IN3, HIGH);
  digitalWrite (IN4, LOW);
  analogWrite(ENB,255);
  delay(2000);
  analogWrite(ENB,107);
  delay(2000);
  analogWrite(ENB,0);
  delay(5000);
}
```

Timer/Contador en Arduino

El microprocesador ATmega328p cuenta con 2 timers/contadores de 8 bit con su preescalador y el modo de comparación, 1 timer/contador de 16 bit con su prescalator, el modo de comparación y el modo de captura, además del oscilador independiente del microprocesador.
El timer 0 y el timer 2 son timers de 8 bits, lo que significa que el valor máximo que pueden contar es 255. El timer 1 es de 16 bits, lo que significa que el valor máximo que puede almacenar es de 65535. Cuando alcanzan su valor máximo, se desborda y vuelve a 0 activándose la interrupción del timer. A 16 MHz, la interrupción en el timer de 8 bits será cada 256/16000000 (16 micros) segundos y 65536/16000000 (4 milis) segundos para el de 16 bits. El tiempo del reloj del ATmega328p es de 16MHz
Para no tener un tiempo de desborde muy bajo, se usa el preescalador que lo que hace es “disminuir la frecuencia”, es decir, hace las operaciones con menor velocidad.
Ejemplo, si se quiere el desborde o la interrupción cada segundo (frecuencia de 1 Hz), sería la frecuencia del reloj entre el preescalador *la frecuencia del tiempo – 1 que es el valor tomando el cero. Esto daría 16000000/(1024*1) – 1 = 15624. Como es mayor a 255, se utilizaría el timer 1 de 16 btis.

Cuando el timer se desborda, el bit del registro TIMSKx se active siempre y cuando el bit del registro TOIEx del timer esté activado. Cuando ocurre esto, será llamada una subrutina en el programa ISR(TIMERx_OVF_vect).

Los registros del timer en el microcontrolador son: TCNTx – Registro del timer/Contador en donde se almacena el valor actual; OCRx – Registro de comparación de salida; ICRx – Registro de captura de entrada (sólo para el timer de 16 bit); TIMSKx – Registro del timer/Contador de interrupción de máscara (para habilitar o deshabilitar las interrupciones del timer); TIFRx – Registro del timer/Contador de la bandera de interrupción (indica si hay interrupción pendiente o activada). (x = lugar del timer 0, 1 ó 2; y después el registro A o B).

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Para calcular la frecuencia o el tiempo en que se debe activar la interrupción del timer es mediante:

- Frecuencia del microprocesador (16MHz Arduino UNO)
- Máximo valor a contar (256 de 8 bit y 65536 para el timer de 16 bit)
- Dividir la frecuencia del microprocesador entre el preescalador que puede ser 8,64,256,1024.
- El resultado dividirlo entre la frecuencia deseada final y si el valor es menor al máximo valor a contar está correcto y es el valor que se carga. De lo contrario, seleccionar otro preescalador de mayor valor.

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

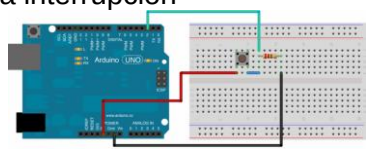
Código ejemplo

```
int i;
void setup() {
  Serial.begin(9600);
  cli();
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1 = 0;
  OCR1A = 62496;
  TCCR1B |= (1 << WGM12);
  TCCR1B |= (1 << CS12) | (1 << CS10);
  TIMSK1 |= (1 << OCIE1A);
  sei();
}
void loop() {
}
ISR(TIMER1_COMPA_vect){
  i = i + 1;
  Serial.println(i);
}
```

Interrupciones

Las interrupciones son acciones que se ejecutan cuando se recibe una señal específica que indica que el programa o lo que se está realizando se interrumpa para hacer otra acción de acuerdo a esa interrupción. Hay interrupciones por timer, contador, externas.
Los pines en Arduino UNO para interrupciones son: Interrupción 0 – pin 2; Interrupción 1 - pin 3.

Diagrama de conexión para una interrupción



Código ejemplo

```
const int buttonPin = 2;
const int ledPin =  13;
volatile int buttonState = 0;
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
  attachInterrupt(0, pin_ISR, CHANGE);
}
void loop() {
}
void pin_ISR() {
  buttonState = digitalRead(buttonPin);
  digitalWrite(ledPin, buttonState);
}
```