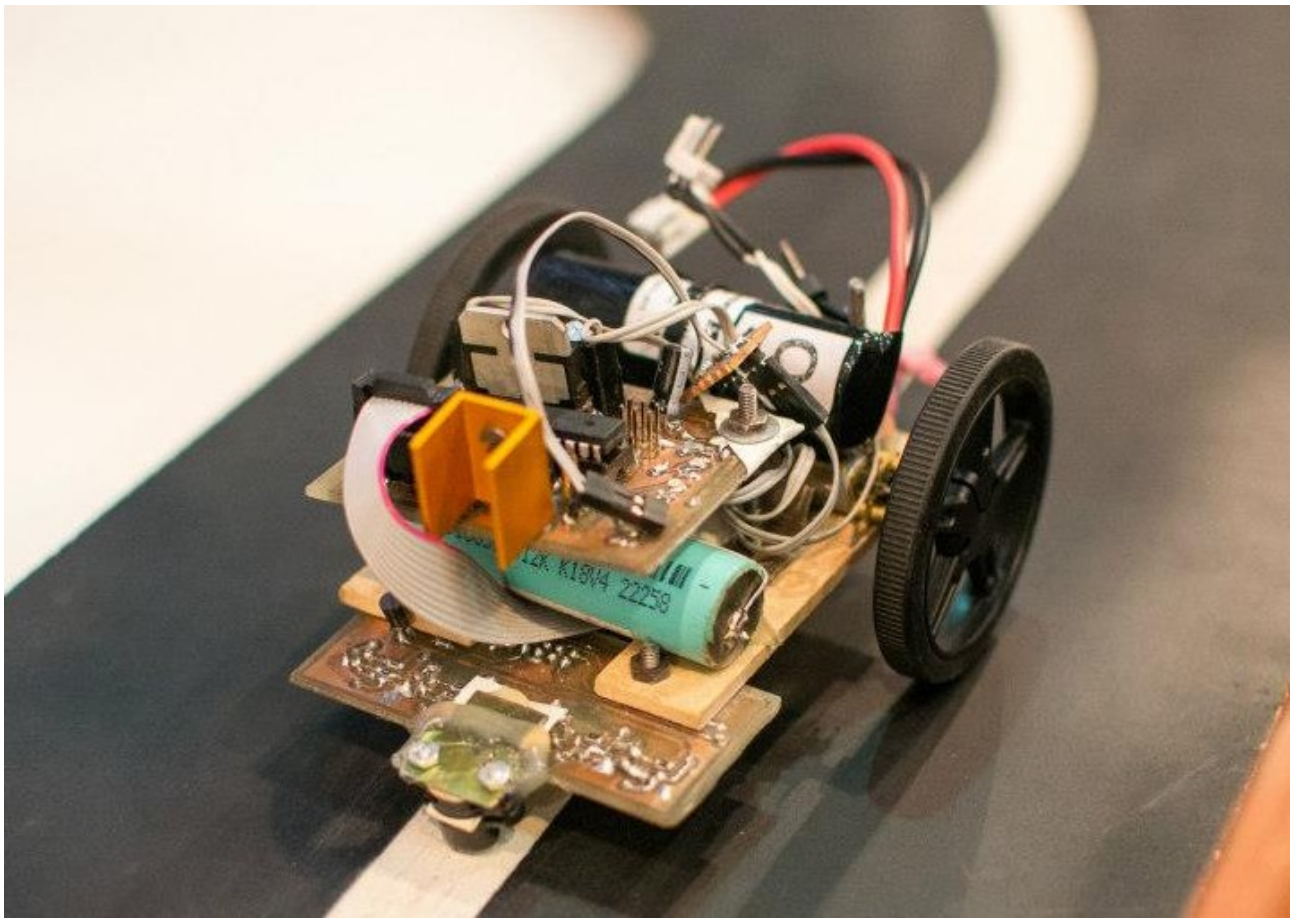
Velocista seguidor de linea CHANCHIS

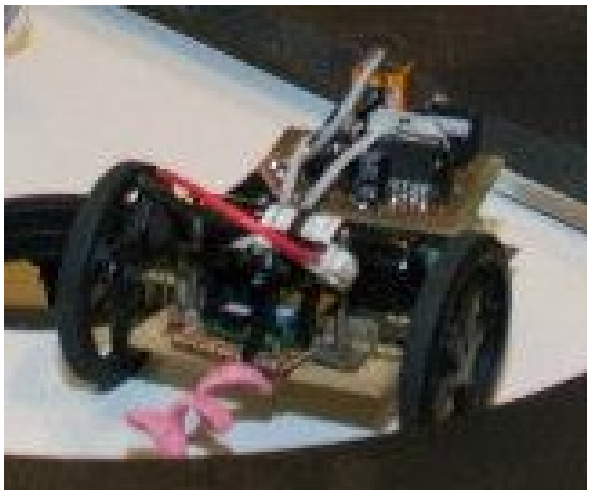
Documentación del robot :

• Descripción de la estructura mecánica del robot :

Chanchis es un robot diseñado para competencias de seguidores de linea, esta hecho en forma de triciclo, una vez que tiene dos ruedas puestas traseras y una bolita adelante que le asegura la estabilidad.

Esta armado sobre una placa de madera atornillada a la placa que contiene los sensores, los motores van puestos acoplados con un enganche de hierro atornillado también a la madera y la placa que lleva el circuito esta adaptada en un tornillo.

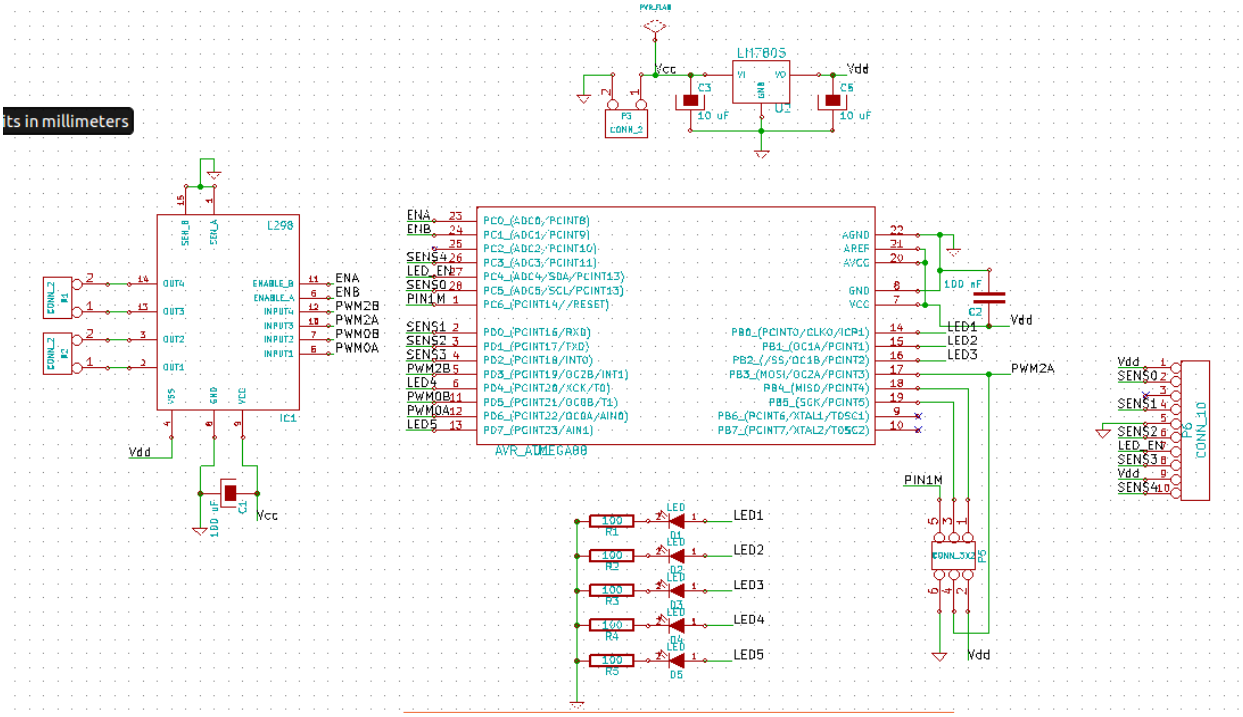


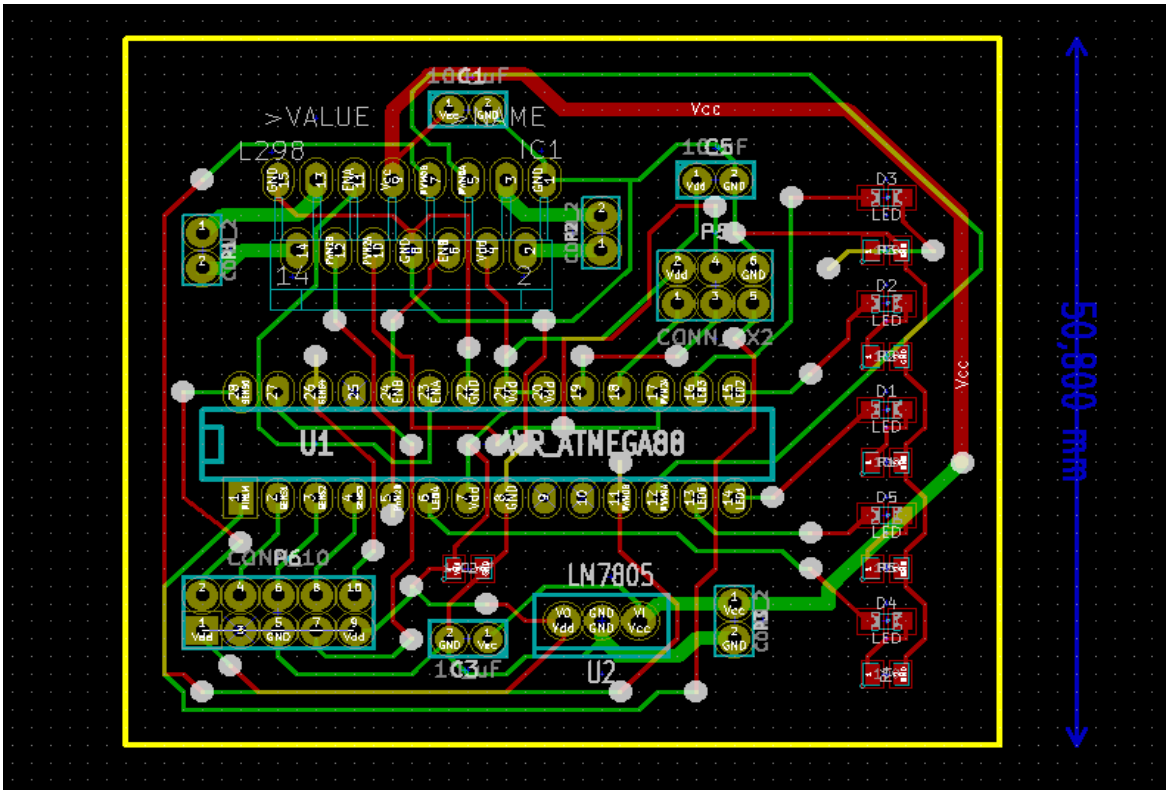


• Descripción de los circuitos electrónicos utilizados:

En el robot usamos para el control el microcontrolador ATMEGA88, con una tensión de 5 voltios, usamos una LM7805 para bajar la tensión de entrada de la batería que tenemos, de 7,2 voltios a partir de dos baterías recargables de 3,6 voltios en serie. También usamos un puente H L298 para el control de los motores.

El robot tiene 5 sensores infrarojos CNY70 para la detección de la linea, dos baterías, y la placa que lleva el circuito.





• Descripción del funcionamiento básico del robot.
Previamente al inicio del seguimiento de la linea, el robot hace una prueba con los sensores tratande de buscar las posiciones que los mismos pueden encontrarse sobre la linea, para tener mejor repuesta de direccionamiento durante el circuito. Se trata de dar vueltas en círculos probando los sensores contra la linea blanca del piso buscando esas posiciones. Luego de dar las vueltas se posiciona el robot en el centro de la linea blanca y el mismo arranca a hacer el circuito. Funciona con control de los dos motores de tracción independiente, aumentando y disminuyendo la velocidad de cada motor para poder hacer curvas y enderezar el robot.

• Código fuente del software

MAKEFILE

```
TARGET = Velocista
MMCU = atmega88
F_CPU = 8000000UL

CC = avr-gcc
CFLAGS = -O -g
SRC = $(shell find .  -name '*.[c]')
INC = $(shell find .  -name '*.[h]')

ifeq ($(MMCU), atmega8)
        TARGET_P = m8
        HFUSE = 0xd9
        LFUSE = 0xe4
        MMCU_N = 0
else
ifeq ($(MMCU), atmega88)
        TARGET_P = m88
        HFUSE = 0xdf
```

```
	LFUSE = 0xe2
	MMCU_N = 1
else
	$(error Target no soportado)
endif
endif

CDEFINES = -DMMCU=$(MMCU_N) -DF_CPU=$(F_CPU)

# Reglas
########

# Reglas para compilar y generar el binario para subir al target
all: hex

bin: $(SRC) $(INC)
	$(CC) $(CDEFINES) -mmcu=$(MMCU) $(CFLAGS) -o $(TARGET).bin $(SRC)
	avr-size -d $(TARGET).bin

hex: bin
	avr-objcopy -j .text -O ihex $(TARGET).bin $(TARGET).hex


# Reglas que tienen sentido si se baja el makefile y se usa localmente.

program: hex
	avrdude -c usbtiny -p $(TARGET_P) -U f:w:$(TARGET).hex -F

program_dw: bin
	avarice -w -j usb --erase --program --file $(TARGET)

fuse:
	avrdude -c usbtiny -p $(TARGET_P) -U lfuse:w:$(LFUSE):m -U hfuse:w:$(HFUSE):m

asm: $(SRC) $(INC)
	$(CC) $(CDEFINES) -mmcu=$(MMCU) $(CFLAGS) -S -o $(TARGET).s $<

clean:
	rm -rf $(TARGET).hex $(TARGET) $(TARGET).s
```

****

CONTROL DE MOTORES

```
#include "motores.h"
void motoresPrender(void)
{
	PORTC |= _BV(PC1); // PWM0
	PORTC |= _BV(PC0); // PWM2
}

void motoresApagar(void)
```

```
{
	PORTC &=~ _BV(PC1); // PWM0
	PORTC &=~ _BV(PC0); // PWM2
}

void cambiarVelocidadMotorIzq(uint8_t velocidad, bool haciaAdelante)
{
	if (haciaAdelante)
	{
		OCR0A = velocidad;
		OCR0B = 0;
	} else {
		OCR0A = 0;
		OCR0B = velocidad;
	}
}

void cambiarVelocidadMotorDer(uint8_t velocidad, bool haciaAdelante)
{
	if (haciaAdelante)
	{
		OCR2A = velocidad;
		OCR2B = 0;
	} else {
		OCR2A = 0;
		OCR2B = velocidad;
	}
}


void motoresVelocidad(int16_t izq, int16_t der)
{
	// Le bajo la velocidad al motor izquierdo porque el derecho anda mas lento
	int16_t velIzqFinal = (izq * 83) / 100;

	if (izq < 0) cambiarVelocidadMotorIzq( (uint8_t)(-velIzqFinal), false);
	else cambiarVelocidadMotorIzq( (uint8_t)(velIzqFinal), true);

	if (der < 0) cambiarVelocidadMotorDer( (uint8_t)(-der), false);
	else cambiarVelocidadMotorDer( (uint8_t)(der), true);
}
```

MOTORES.H

```
#ifndef _MOTORES_H
#define _MOTORES_H

#include "startup.h"
#include "stdbool.h"

void motoresPrender(void);
```

```
void motoresApagar(void);

void motoresVelocidad(int16_t izq, int16_t der);

#endif //_MOTORES_H
```

****

CONTROL DE LOS SENSORES

```
#include "sensores.h"

uint16_t sensoresMaxOn[5] = {ValorProhibido, ValorProhibido, ValorProhibido, ValorProhibido,
ValorProhibido};
uint16_t sensoresMinOn[5] = {ValorProhibido, ValorProhibido, ValorProhibido, ValorProhibido,
ValorProhibido};
uint16_t sensoresMaxOff[5] = {ValorProhibido, ValorProhibido, ValorProhibido, ValorProhibido,
ValorProhibido};
uint16_t sensoresMinOff[5] = {ValorProhibido, ValorProhibido, ValorProhibido, ValorProhibido,
ValorProhibido};

uint8_t mascaraC = 0x00;
uint8_t mascaraD = 0x00;


void emisorPrender(void)
{
	PORTC |= _BV(PC4);
}

void emisorApagar(void)
{
	PORTC &=~ _BV(PC4);
}

void sensoresOutputHigh(void)
{
	DDRC |= mascaraC;
	DDRD |= mascaraD;

	PORTC |= mascaraC;
	PORTD |= mascaraD;
}

void sensoresInput(void)
{
	DDRC &=~ mascaraC;
	DDRD &=~ mascaraD;

	PORTC &=~ mascaraC;
	PORTD &=~ mascaraD;
```

```
}

uint8_t sensoresGetEstadoActualPINC(void)
{
	return (PINC & mascaraC);
}

uint8_t sensoresGetEstadoActualPIND(void)
{
	return (PIND & mascaraD);
}

void sensoresLeer(uint16_t *sensoresValor)
{
	uint8_t i;
	uint8_t start_time;
	uint8_t delta_time;
	uint16_t time = 1; // Para saber si le asigne o no un valor mas adelante
	bool cambioSensor[5] = {false, false, false, false, false};

	for(i = 0; i < 5; i++)
		sensoresValor[i] = 0;

	sensoresOutputHigh();
	_delay_us(10);

	sensoresInput();


	uint8_t ultimoPINC = mascaraC;
	uint8_t ultimoPIND = mascaraD;

	uint8_t prevTCCR2A = TCCR2A;
	uint8_t prevTCCR2B = TCCR2B;

	start_time = TCNT2;
	while (time < TimeOutSensores)
	{
		delta_time = TCNT2 - start_time;
		time += delta_time;
		start_time += delta_time;

		if (sensoresGetEstadoActualPINC() == ultimoPINC &&
			sensoresGetEstadoActualPIND() == ultimoPIND)
			continue;

		// Guardo los ultimos valores observados
		ultimoPINC = sensoresGetEstadoActualPINC();
		ultimoPIND = sensoresGetEstadoActualPIND();

		// Me fijo cual estado cambio
		if ( ((ultimoPINC & _BV(PC5)) == 0) && (cambioSensor[0] == false) )
```

```
                {
                        sensoresValor[0] = time;
                        cambioSensor[0] = true;
                } else if ( ((ultimoPIND & _BV(PD0)) == 0) && (cambioSensor[1] == false) )
                {
                        sensoresValor[1] = time;
                        cambioSensor[1] = true;
                } else if ( ((ultimoPIND & _BV(PD1)) == 0) && (cambioSensor[2] == false) )
                {
                        sensoresValor[2] = time;
                        cambioSensor[2] = true;
                } else if ( ((ultimoPIND & _BV(PD2)) == 0) && (cambioSensor[3] == false) )
                {
                        sensoresValor[3] = time;
                        cambioSensor[3] = true;
                } else if ( ((ultimoPINC & _BV(PC3)) == 0) && (cambioSensor[4] == false) )
                {
                        sensoresValor[4] = time;
                        cambioSensor[4] = true;
                }
        }

        for(i = 0; i < 5; i++)
                if (sensoresValor[i] == 0)
                        sensoresValor[i] = TimeOutSensores;
}

void sensoresLeerConCorrecion(uint16_t *sensoresValor)
/* Hago una medicion con los leds de los sensores prendidos y luego otra
 * con los sensores apagados, luego descuento la medicion con los leds
 * apagados a la medicion con los leds encendidos. */
{
        uint16_t sensoresApagado[5];
        uint8_t i;

        emisorApagar();
        sensoresLeer(sensoresApagado);

        emisorPrender();
        sensoresLeer(sensoresValor);

        for (i = 0; i < 5; i++)
        {
                sensoresValor[i] += TimeOutSensores - sensoresApagado[i];
        }
}

void sensoresLeerCalibrado(uint16_t *sensoresValor)
// Convierte a los valores al rango 0-1000
{
        uint8_t i;
```

```
	sensoresLeerConCorrecion(sensoresValor);

	for (i = 0; i < 5; i++)
	{
		uint16_t calmin,calmax;
		uint16_t denominador;

		if(sensoresMinOff[i] < sensoresMinOn[i]) // senial no significativa
			calmin = TimeOutSensores;
		else
			calmin = sensoresMinOn[i] + TimeOutSensores - sensoresMinOff[i];

		if(sensoresMaxOff[i] < sensoresMaxOn[i]) // senial no significativa
			calmax = TimeOutSensores;
		else
			calmax = sensoresMaxOn[i] + TimeOutSensores - sensoresMaxOff[i];


		denominador = calmax - calmin;

		int16_t x = 0;
		if(denominador != 0)
			x = (((int32_t)sensoresValor[i]) - calmin) * 1000 / denominador;

		if(x < 0)
			x = 0;
		else if(x > 1000)
			x = 1000;

		// 0 para negro, 1000 para blanco.
		sensoresValor[i] = 1000 - x;
	}

}

uint16_t leerLinea(void)
{
	uint8_t i, sobreLinea = 0;
	uint32_t promedio = 0;
	uint16_t sum = 0;
	static uint16_t ultimoValor = 0; // Se asume inicialmente que la linea esta a la izquierda
	uint16_t sensoresValor[5];

	sensoresLeerCalibrado(sensoresValor);


	for (i = 0; i < 5; i++) {
		int16_t valor = sensoresValor[i];

		if(valor > MargenLineaBlanca) {
			sobreLinea = 1;
		}
```

```
			// Solo se usan los valores que estan sobre un margen de ruido
			if(valor > MargenRuido) {
				promedio += (int32_t)(valor) * (i * 1000);
				sum += valor;
			}
		}

		if(!sobreLinea)
		{
			// Si la ultima lectura fue del medio a la izquierda
			if(ultimoValor < (5-1)*1000/2)
				return 0;
			else
				return (5-1)*1000;

		}

		ultimoValor = promedio/sum;

		return ultimoValor;
}

void obtenerValoresCalibracion(uint16_t *sensoresMin, uint16_t *sensoresMax)
{
		uint8_t i, j;
		uint16_t sensoresTemp[5];


		// Para la primer medicion se le asigna el mismo valor a ambas variables
		for (j = 0; j < 10; j++)
		{
			sensoresLeer(sensoresTemp);
			for (i = 0; i < 5; i++)
			{
				if (sensoresTemp[i] < sensoresMin[i] || sensoresMin[i] == ValorProhibido)
					sensoresMin[i] = sensoresTemp[i];

				if (sensoresTemp[i] > sensoresMax[i] || sensoresMax[i] == ValorProhibido)
					sensoresMax[i] = sensoresTemp[i];
			}
		}
}

uint8_t sensoresLeerDigital(uint16_t lineaBlancaLimite)
{
		uint8_t i;
		uint8_t sensoresDigital = 0x00;
		uint16_t sensoresValor[5];

		sensoresLeerConCorrecion(sensoresValor);
```

```
	if (sensoresValor[0] < lineaBlancaLimite) sensoresDigital |= _BV(SENS0);
	if (sensoresValor[1] < lineaBlancaLimite) sensoresDigital |= _BV(SENS1);
	if (sensoresValor[2] < lineaBlancaLimite) sensoresDigital |= _BV(SENS2);
	if (sensoresValor[3] < lineaBlancaLimite) sensoresDigital |= _BV(SENS3);
	if (sensoresValor[4] < lineaBlancaLimite) sensoresDigital |= _BV(SENS4);

	return sensoresDigital;
}

uint8_t sensoresLeerDigitalConCalibracion(void)
{
	uint8_t i;
	uint8_t sensoresDigital = 0x00;
	uint16_t sensoresValor[5];

	sensoresLeerCalibrado(sensoresValor);

	if (sensoresValor[0] > MargenLineaBlanca) sensoresDigital |= _BV(SENS0);
	if (sensoresValor[1] > MargenLineaBlanca) sensoresDigital |= _BV(SENS1);
	if (sensoresValor[2] > MargenLineaBlanca) sensoresDigital |= _BV(SENS2);
	if (sensoresValor[3] > MargenLineaBlanca) sensoresDigital |= _BV(SENS3);
	if (sensoresValor[4] > MargenLineaBlanca) sensoresDigital |= _BV(SENS4);

	return sensoresDigital;
}

void actualizarValoresCalibracion(void)
{
	emisorPrender();
	obtenerValoresCalibracion(sensoresMinOn, sensoresMaxOn);

	emisorApagar();
	obtenerValoresCalibracion(sensoresMinOff, sensoresMaxOff);
}

void sensoresStartup(void)
{
	mascaraC |= _BV(PC3);
	mascaraC |= _BV(PC5);

	mascaraD |= _BV(PD0);
	mascaraD |= _BV(PD1);
	mascaraD |= _BV(PD2);
}
```

****

SENSORES.H

```
#ifndef _SENSORES_H
#define _SENSORES_H

#include <avr/io.h>
#include <util/delay.h>
```

```
#include <stdbool.h>

#define ValorProhibido		10000
#define SENS0		0
#define SENS1		1
#define SENS2		2
#define SENS3		3
#define SENS4		4

#define TimeOutSensores		5000	// Ciclos que va a contar hasta considerar
                                                                        // al sensor esta sobre negro.
#define MargenRuido		50
#define MargenLineaBlanca	200


//		PC5 Sensor <<
//		PD0 Sensor <
//		PD1 Sensor ||
//		PD2 Sensor >
//		PC3 Sensor >>


// Se debe ejecutar antes de llamar a los metodos de esta biblioteca.
void sensoresStartup(void);

// Devuelve un uint8_t donde estan mapeados los sensores, en el bit SENSX
// esta la lectura del sensor X.
uint8_t sensoresLeerDigital(uint16_t lineaBlancaLimite);

// Devuelve un uint8_t donde estan mapeados los sensores, en el bit SENSX
// esta la lectura del sensor X.
// PRE: Se deben calibrar los sensores antes de usar esta funcion
uint8_t sensoresLeerDigitalConCalibracion(void);

// Devuelve un valor en el rango 0-4000 que indica que corrimiento
// tiene sobre la linea.
uint16_t leerLinea(void);

// Obtiene los valores con los que va a calcular las mediciones calibradas
// Se debe ejecutar antes de obtener alguna medicion.
void actualizarValoresCalibracion(void);

#endif // _SENSORES_H
```

****

STARTUP

```
#include "startup.h"

void configurarEntradas(void)
{
	//		DDRB &=~ _BV(PB0);			// Boton para empezar
```

```
	//		PORTB |= _BV(PB0);
}

void configurarPinesMotor(void)
{
	// Tengo un PWM por cada motor

	//PWM 0
	DDRD |= _BV(PD5); // B
	DDRD |= _BV(PD6); // A
	PORTD &=~ _BV(PD5);
	PORTD &=~ _BV(PD6);

	//PWM 2
	DDRD |= _BV(PD3); // B
	DDRB |= _BV(PB3); // A
	PORTD &=~ _BV(PD3);
	PORTB &=~ _BV(PB3);


	DDRC |= _BV(PC1); // Enable PWM 0
	DDRC |= _BV(PC0); // Enable PWM 2
	PORTC &=~ _BV(PC1);
	PORTC &=~ _BV(PC0);
}

void configurarSalidasLeds(void)
{
	// De izquierda a derecha

	DDRD |= _BV(PD4);
	DDRD |= _BV(PD7);
	DDRB |= _BV(PB0);
	DDRB |= _BV(PB1);
	DDRB |= _BV(PB2);
	PORTD &=~ _BV(PD4);
	PORTD &=~ _BV(PD7);
	PORTB &=~ _BV(PB0);
	PORTB &=~ _BV(PB1);
	PORTB &=~ _BV(PB2);
}

void configurarSalidas(void)
{
	DDRC |= _BV(PC4); // Leds de sensores

	configurarSalidasLeds();
	configurarPinesMotor();
}

void cambiarFrecuenciaMicro(void)
{
```

```
	CLKPR = (1<<CLKPCE);    // Necesario para poder cambiar el prescaler del micro
	CLKPR = 0;              // Prescaler 1, 8 Mhz
	_delay_ms(50);
}


void PWMMotorStartup(void)
{
	// PWM 0

	// Fast PWM Mode, inverting mode
	TCCR0A &=~ _BV(COM0A0);
	TCCR0A |= _BV(COM0A1);
	TCCR0A &=~ _BV(COM0B0);
	TCCR0A |= _BV(COM0B1);
	// Fast PWM, WGMode 3
	TCCR0A |= _BV(WGM00);
	TCCR0A |= _BV(WGM01);
	TCCR0B &=~ _BV(WGM02);
	// No prescaling
	TCCR0B |= _BV(CS00);
	TCCR0B &=~ _BV(CS01);
	TCCR0B &=~ _BV(CS02);
	// Set output to 0
	OCR0A = OCR0B = 0;


	// PWM 2

	// Fast PWM Mode, inverting mode
	TCCR2A &=~ _BV(COM2A0);
	TCCR2A |= _BV(COM2A1);
	TCCR2A &=~ _BV(COM2B0);
	TCCR2A |= _BV(COM2B1);
	// Fast PWM, WGMode 3
	TCCR2A |= _BV(WGM00);
	TCCR2A |= _BV(WGM01);
	TCCR2B &=~ _BV(WGM02);
	// No prescaling
	TCCR2B |= _BV(CS20);
	TCCR2B &=~ _BV(CS21);
	TCCR2B &=~ _BV(CS22);
	// Set output to 0
	OCR2A = OCR2B = 0;
}

void startup(void)
{
	configurarEntradas();
	configurarSalidas();

	cambiarFrecuenciaMicro();
```

```
	PWMMotorStartup();

	// Los sensores se configuran mas adelante en el programa.
}
```

****

STARTUP.H

```
#ifndef _STARTUP_H
#define _STARTUP_H

#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

void startup(void);

uint8_t getMascaraSensores(void);

#endif //_STARTUP_H
```

****

VELOCISTA.C

```
#include <stdbool.h>
#include "startup.h"
#include "motores.h"
#include "sensores.h"

#define LED1ON	PORTD |= _BV(PD4)
#define LED2ON	PORTD |= _BV(PD7)
#define LED3ON	PORTB |= _BV(PB0)
#define LED4ON	PORTB |= _BV(PB1)
#define LED5ON	PORTB |= _BV(PB2)

#define LED1OFF	PORTD &=~ _BV(PD4)
#define LED2OFF	PORTD &=~ _BV(PD7)
#define LED3OFF	PORTB &=~ _BV(PB0)
#define LED4OFF	PORTB &=~ _BV(PB1)
#define LED5OFF	PORTB &=~ _BV(PB2)

#define Kp	/3
#define Kd	8
#define Ki	/1000

#define VelocidadMaxima	255

void apagarLeds(void)
{
	LED1OFF;
	LED2OFF;
	LED3OFF;
```

```
	LED4OFF;
	LED5OFF;
}

void mostrarSensoresDesdeDatos(uint8_t sensoresDigital)
{
	apagarLeds();

	if ( (sensoresDigital & _BV(SENS0)) != 0) LED1ON;
	if ( (sensoresDigital & _BV(SENS1)) != 0) LED2ON;
	if ( (sensoresDigital & _BV(SENS2)) != 0) LED3ON;
	if ( (sensoresDigital & _BV(SENS3)) != 0) LED4ON;
	if ( (sensoresDigital & _BV(SENS4)) != 0) LED5ON;
}

void mostrarSensores()
{
	uint8_t sensoresDigital;

	while(1) mostrarSensoresDesdeDatos( sensoresLeerDigital(700) );

}

void mostrarSensoresCalibrados()
{
	uint8_t sensoresDigital;

	while(1) mostrarSensoresDesdeDatos( sensoresLeerDigitalConCalibracion() );

}

void inicializarSinMover(void)
{
	uint8_t i;

	for (i = 0; i < 150; i++)
	{
		PORTB |= _BV(PB0);

		actualizarValoresCalibracion();

		_delay_ms(10);
		PORTB &=~ _BV(PB0);
		_delay_ms(1);
	}

}

void pulsadorArranque()
{
	bool comenzar = false;
```

```
	while(1)
	{
		mostrarSensores();
		if (~PINB & (1<<PB0))
		{
			comenzar = true;
			continue;
		}
		if (comenzar) break;
	}
}

void ledsTest(void)
{
	const uint8_t vel = 75;

	while(1)
	{
		PORTB &=~ _BV(PB2);
		PORTD |= _BV(PD4);
		_delay_ms(vel);
		PORTD &=~ _BV(PD4);
		PORTD |= _BV(PD7);
		_delay_ms(vel);
		PORTD &=~ _BV(PD7);
		PORTB |= _BV(PB0);
		_delay_ms(vel);
		PORTB &=~ _BV(PB0);
		PORTB |= _BV(PB1);
		_delay_ms(vel);
		PORTB &=~ _BV(PB1);
		PORTB |= _BV(PB2);
		_delay_ms(vel);
	}
}

void mostrarLedsInicializar(uint16_t i, uint16_t maximo)
{
	uint16_t paso = maximo/9;

	apagarLeds();

	if (i > 0*paso && i < 5*paso) LED1ON;
	if (i > 1*paso && i < 6*paso) LED2ON;
	if (i > 2*paso && i < 7*paso) LED3ON;
	if (i > 3*paso && i < 8*paso) LED4ON;
	if (i > 4*paso && i < 9*paso) LED5ON;

}

void inicializar(void)
{
```

```
	const uint16_t velocidad = 100;
	const uint16_t tiempoInicializacion = 150;
	uint16_t i;

	motoresPrender();

	for (i = 0; i < tiempoInicializacion; i++)
	{
		mostrarLedsInicializar(i, tiempoInicializacion);

		motoresVelocidad(velocidad, -velocidad);

		actualizarValoresCalibracion();

		_delay_ms(10);
	}

	motoresVelocidad(0, 0);

	motoresApagar();

	apagarLeds();
}

void arranqueDespacio(void)
{
	uint16_t vel;
	uint8_t i;
	const uint8_t cantidadIncrementos = 80;

	for (i = 20; i < cantidadIncrementos; i++)
	{
		vel = (i*VelocidadMaxima)/cantidadIncrementos;
		motoresVelocidad( (uint8_t) vel, (uint8_t) vel);
		_delay_ms(5);
	}

}

int main(void)
{
	uint16_t posicion;

	int16_t proporcionalAnterior = 0;
	int16_t proporcional;
	int16_t derivativo;
	int32_t integral = 0;

	// Guarda la diferencia entre las velocidades de los motores.
	int16_t diferenciaVelocidades;

	startup();
```

```
	sensoresStartup();

	//pulsadorArranque();

	_delay_ms(2500);
	inicializar();
	_delay_ms(4500);

	motoresPrender();

	arranqueDespacio();

	while(1)
	{
		posicion = leerLinea();

		proporcional = (int16_t) posicion - 2000; // Ya que 2000 es el centro
		derivativo = proporcional - proporcionalAnterior;
		integral += proporcional;

		proporcionalAnterior = proporcional;

		diferenciaVelocidades = proporcional Kp + derivativo*Kd + integral Ki;

		if (diferenciaVelocidades > VelocidadMaxima)
			diferenciaVelocidades = VelocidadMaxima;
		else if (diferenciaVelocidades < -VelocidadMaxima)
			diferenciaVelocidades = -VelocidadMaxima;

		if (diferenciaVelocidades < 0)
			motoresVelocidad(VelocidadMaxima + diferenciaVelocidades,
VelocidadMaxima);
		else
			motoresVelocidad(VelocidadMaxima, VelocidadMaxima -
diferenciaVelocidades);
	}

	return 0;
}
```

• Lista de componentes y costo de construcción del robot.

COMPONENTE:	**VALOR APROX:**
2 Motores Pololu	$159,50*
1 Placa de circuito impreso de cobre	$10,00
1 Atmega 88	$30,00
1 L298	$32,89
5 sensores CNY70	$150,00

2 Ruedas Polol----------$79,50*
1 LM7805----------$25,00
Leds varios----------$5,00
Resistencias Varias----------$2,00
Zócalos Varios----------$5,00
Capacitores Varios----------$5,00
2 Pilas recargables----------$40,00
Cables varios----------$5,00
Bolita Pololu ----------$19,95*

TOTAL:
$568,84**

*valores sacados del sitio http://www.pololu.com en dólares, cambio de 1 U$ = 5 pesos argentinos.
**Valor en pesos argentinos.