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

#ifndef F_CPU
//define cpu clock speed if not defined
#define F_CPU 16000000UL
#endif

//set desired baud rate
#define BAUDRATE 1200

//calculate UBRR value
#define UBRRVAL ((F_CPU/(BAUDRATE*16UL))-1)

//define receive parameters
#define SYNC 0XAA// synchro signal
#define RADDR 0xBB

#define LEDON 0x11//LED on command
#define LEDOFF 0x22//LED off command

#define TURN_LEFT 0x00
#define STOP_TURN_LEFT 0x01

#define TURN_RIGHT 0x02
#define STOP_TURN_RIGHT 0x03

#define MOVE_FORWARD 0x04
#define STOP_MOVE_FORWARD 0x05

#define MOVE_BACKWARD 0x06
#define STOP_MOVE_BACKWARD 0x07

#define FORWARD 0
#define BACKWARD 1
#define LEFT 2
#define RIGHT 3

#define SAFETY_FORWARD 4
#define SAFETY_BACKWARD 5

#define SAFETY 6

#define STATIONARY 6
```

```
#define ITERATIONS 10
#define WAIT 100

// Finite state machine
int status = 6;

/* Sensor variables */
int burstCycle = 0;
int burstPause = 0;
int cycle = 0;

void turnLeft(void )
{
// PORTA bits 2 and 7 involved
PORTA |= (1<<7); // Pin 2 On
PORTA &= ~(1<<2); // Pin 7 Off
}
void stopTurnLeft(void ) {
PORTA &= ~ ((1 << PA2) | (1 << PA7));
}

void turnRight(void )
{
// PORTA bits 2 and 7 involved
PORTA |= (1<<2); // Pin 7 On
PORTA &= ~(1<<7); // Pin 2 Off
}
void stopTurnRight(void ) {
PORTA &= ~ ((1 << PA2) | (1 << PA7));
}

void moveForward(void )
{
// PORTA bits 0 and 1 involved
PORTA |= (1<<PA1); // Pin 1 On
PORTA &=~(1<<PA0); // Pin 0 Off
// Pin 1 Off
}
void stopMoveForward(void ) {
PORTA &= ~((1 << PA0) | (1 << PA1));
}

void moveBackward(void )
{
```

```
// PORTA bits 0 and 1 involved
PORTA &=~(1<<PA1); // Pin 1 Off
PORTA |= (1<<PA0); // Pin 0 On
}
void stopMoveBackward(void ) {
PORTA &= ~((1 << PA0) | (1 << PA1));
}

void USART_Init(void)
{

//Set baud rate
UBRRL=(uint8_t)UBRRVAL; //low byte
UBRRH=(UBRRVAL>>8); //high byte

//Set data frame format: asynchronous mode,no parity, 1 stop bit, 8 bit size
UCSRC = (1<<URSEL) | (0<<UMSEL) | (0<<UPM1) | (0<<UPM0) |
(0<<USBS)  | (0<<UCSZ2) | (1<<UCSZ1)| (1<<UCSZ0);

//Enable Transmitter and Receiver and Interrupt on receive complete
UCSRB=(1<<RXEN)|(1<<RXCIE);
}

uint8_t USART_vReceiveByte(void)
{
   // Wait until a byte has been received
   while((UCSRA&(1<<RXC)) == 0);

   // Return received data
   return UDR;
}

ISR(USART_RXC_vect)
{
//define variables
uint8_t raddress, data, chk;//transmitter address

//receive destination address
raddress=USART_vReceiveByte();
//receive data
data=USART_vReceiveByte();
//receive checksum
chk=USART_vReceiveByte();

//compare received checksum with calculated
```

```
if(chk==(raddress+data))//if match perform operations
{
//if transmitter address match
if(raddress==RADDR)
{
if(data == MOVE_FORWARD) {
status = FORWARD;
moveForward();
}
else if (data == STOP_MOVE_FORWARD) {
stopMoveForward();
status = STATIONARY;
}
else if(data == MOVE_BACKWARD) {
status = BACKWARD;
moveBackward();
}
else if (data == STOP_MOVE_BACKWARD) {
status = STATIONARY;
stopMoveBackward();

}
else if(data == TURN_LEFT) {
status = LEFT;
moveForward();
turnLeft();
}
else if (data == STOP_TURN_LEFT) {
stopMoveForward();
stopTurnLeft();
status = STATIONARY;
}
else if(data == TURN_RIGHT) {
status = RIGHT;
moveForward();
turnRight();
}
else if (data == STOP_TURN_RIGHT) {
stopMoveForward();
stopTurnRight();
status = STATIONARY;
}
}
}
}
```

```
void Main_Init(void)
{

// Enabling port B for the receiver module
DDRB = 0xFF;
PORTB |= (1 << PB5); // Only PB5 on

// This is for the motors
DDRA = 0xFF;
PORTA = 0x00; // Initially OFF

// This is for the led, initially OFF
// Pin 7 is output pin
DDRD |= (1 << PD7);

// Pins 5 and 6 are input pins
DDRD &= ~((1 << PD4) | (1 << PD5) | (1 << PD6));

// Set up pull-ups
PORTD |= ((1 << PD4) | (1 << PD5) | (1 << PD6));

PORTD &= ~(1 << PD7); // Initially OFF

// Start interrupts
sei();
}

void Init_Frequence_Generator()
{
TCCR1B |= (1 << WGM12); // Configure timer 1 for CTC mode

TIMSK  |= (1 << OCIE1A); // Enable CTC interrupt

// 421 = (1/38khz) / (1 / 16Mhz)
OCR1A = 421; // Set CTC compare value to 38KHz at 16Mhz AVR clock, with a prescaler of 1

// Set up timer
TCCR1B |= (1 << CS10); // Start timer at FCPU/1

cycle = 1;
}

int main(void)
```

```
{

int i;

Init_Frequence_Generator();

Main_Init();

USART_Init();

for (;;) {
/*
if ((PIND & 0x20) == 0x00) {

PORTA = 0x00;
PORTD |= (1 << 7);
status = SAFETY;

moveBackward();
for (i = 0; i < ITERATIONS; i++)
_delay_ms(WAIT/2);
} else {
if (status == SAFETY) {
stopMoveBackward();
PORTD &= ~(1 << 7);
status = STATIONARY;
}
}


if ((PIND & 0x40) == 0x00) {

PORTA = 0x00;
PORTD |= (1 << 7);
status = SAFETY;

moveForward();
for (i = 0; i < ITERATIONS; i++)
_delay_ms(WAIT/2);
} else {
if (status == SAFETY) {
stopMoveForward();
PORTD &= ~(1 << 7);
status = STATIONARY;
}
}
*/
if ((PIND & 0x10) == 0) {
PORTD |= (1 << 7);
} else {
PORTD &= ~(1 << 7);
}
}

return 0;

}

// Interrupt code for the infrared light.
ISR(TIMER1_COMPA_vect) {

if (cycle)
{
// Toggle the infrared LED
PORTD ^= ((1 << PB0) | (1 << PB1) | (1 << PB2));

burstCycle++;
if (burstCycle >= 15)
{
burstCycle = 0;
cycle = 0;
PORTD &= ~((1 << PB0) | (1 << PB1) | (1 << PB2));
}
}
else
{
// There have to be some breaks from time to time
burstPause++;
if (burstPause >= 15)
{
burstPause = 0;
cycle = 1;
}
}

}
```