```
#include <ros.h> //dedicada à comunicação dos pacotes ROS da
bib. rosserial-arduino
#include <ros/time.h> //responsável pela sincronização dos
tempos ROS da bib. rosserial-arduino
#include <geometry_msgs/Twist.h> //trata vetores lineares a
angulares ROS da bib. rosserial-arduino
#include <sensor_msgs/Range.h> //função ROS que trada da
distância sonares da bib. rosserial-arduino

#include <NewPing.h> //biblioteca dedicada ao uso de sonares
#include <SimpleKalmanFilter.h> //biblioteca de filtros
digitais

#define IN1 10 //define que o pino 10 receberá o valor da
variável IN1
#define IN2 11 //define que o pino 11 receberá o valor da
variável IN2
#define IN3 12 //define que o pino 12 receberá o valor da
variável IN3
#define IN4 13 //define que o pino 13 receberá o valor da
variável IN4

#define SONAR_NUM 4          //Número de sonares
#define MAX_DISTANCE 200     //Maxima distância de detecção
dos obstáculos
#define PING_INTERVAL 33     //Loop do ping a cada 33
microsecondos.

unsigned long pingTimer[SONAR_NUM];     //tempo de ping para
cada sensor.
unsigned int cm[SONAR_NUM];             // Onde as
distâncias de ping são armazenadas.
uint8_t currentSensor = 0;              // Mantém o controle
de qual sensor está ativo.
unsigned long _timerStart = 0;          //tempo inicia em 0
int LOOPING = 40;                       //Loop a cada 40
milissegundos.
uint8_t oldSensorReading[3];            //Armazena o último
valor válido dos sensores.

uint8_t oneSensor;         //Armazena o valor real do sensor
1.
uint8_t twoSensor;         //Armazena o valor real do sensor
2.
uint8_t threeSensor;       //Armazena o valor real do sensor
3.
```

```
uint8_t fourSensor;         //Armazena o valor real do sensor
4.
uint8_t fiveSensor;
uint8_t sixSensor;
uint8_t sevenSensor;
uint8_t eightSensor;

uint8_t oneSensorKalman;    //Armazena o valor filtrado do
sensor 1.
uint8_t twoSensorKalman;    //Armazena o valor filtrado do
sensor 2.
uint8_t threeSensorKalman;  //Armazena o valor filtrado do
sensor 3.
uint8_t fourSensorKalman;   //Armazena o valor filtrado do
sensor 4.
uint8_t fiveSensorKalman;
uint8_t sixSensorKalman;
uint8_t sevenSensorKalman;
uint8_t eightSensorKalman;


NewPing sonar[SONAR_NUM] = {
  NewPing(3, 2, MAX_DISTANCE), //Pino de Trigger, Pino de
echo, distância máxima.
  NewPing(5, 4, MAX_DISTANCE),
  NewPing(7, 6, MAX_DISTANCE),
  NewPing(9, 8, MAX_DISTANCE)
//  NewPing(15, 14, MAX_DISTANCE),
//  NewPing(17, 16, MAX_DISTANCE),
//  NewPing(19, 18, MAX_DISTANCE),
//  NewPing(21, 20, MAX_DISTANCE)

};

SimpleKalmanFilter KF_1(2, 2, 0.01); //(incerteza medição,
incerteza estimada, ruído);
SimpleKalmanFilter KF_2(2, 2, 0.01);
SimpleKalmanFilter KF_3(2, 2, 0.01);
SimpleKalmanFilter KF_4(2, 2, 0.01);
//SimpleKalmanFilter KF_5(2, 2, 0.01);
//SimpleKalmanFilter KF_6(2, 2, 0.01);
//SimpleKalmanFilter KF_7(2, 2, 0.01);
//SimpleKalmanFilter KF_8(2, 2, 0.01);

ros::NodeHandle nh; //iniciará o nó na placa Arduino.
```

```
//loop em todos os sensores
void sensorCycle() {
  for (uint8_t i = 0; i < SONAR_NUM; i++) {
    if (millis() >= pingTimer[i]) {
      pingTimer[i] += PING_INTERVAL * SONAR_NUM;
      if (i == 0 && currentSensor == SONAR_NUM - 1)
oneSensorCycle();
     sonar[currentSensor].timer_stop();
      currentSensor = i;
      cm[currentSensor] = 0;
     sonar[currentSensor].ping_timer(echoCheck);
    }
  }
}

void echoCheck() // Se o ping for recebido, defina a
distância do sensor para array.
{
 if (sonar[currentSensor].check_timer())
   cm[currentSensor] = sonar[currentSensor].ping_result /
US_ROUNDTRIP_CM;
}

void oneSensorCycle() // Retorna o último valor válido do
sensor.
{
  oneSensor =   returnLastValidRead(0, cm[0]);
  twoSensor =   returnLastValidRead(1, cm[1]);
  threeSensor = returnLastValidRead(2, cm[2]);
  fourSensor =  returnLastValidRead(3, cm[3]);
//  fiveSensor =  returnLastValidRead(4, cm[4]);
//  sixSensor =   returnLastValidRead(5, cm[5]);
//  sevenSensor = returnLastValidRead(6, cm[6]);
//  eightSensor = returnLastValidRead(7, cm[7]);
}

//Se o valor do sensor for 0, então retorna o último valor
armazenado diferente de 0.
int returnLastValidRead(uint8_t sensorArray, uint8_t cm) {
  if (cm != 0) {
    return oldSensorReading[sensorArray] = cm;
  } else {
    return oldSensorReading[sensorArray];
  }
}
```

```
void applyKF() //Aplica o Filtro Kalman na leitura do sensor.
{
  oneSensorKalman   = KF_1.updateEstimate(oneSensor);
  twoSensorKalman   = KF_2.updateEstimate(twoSensor);
 threeSensorKalman = KF_3.updateEstimate(threeSensor);
  fourSensorKalman  = KF_4.updateEstimate(fourSensor);
//  fiveSensorKalman  = KF_5.updateEstimate(fiveSensor);
//  sixSensorKalman   = KF_6.updateEstimate(sixSensor);
// sevenSensorKalman = KF_7.updateEstimate(sevenSensor);
// eightSensorKalman = KF_8.updateEstimate(eightSensor);
}

void startTimer() //a função para começar a contar o tempo
usando millis()
{
  _timerStart = millis();
}

bool isTimeForLoop(int _mSec)  //Verifica se o tempo passou e
retorna true.
{
  return (millis() - _timerStart) > _mSec;
}

void sensor_msg_init(sensor_msgs::Range &range_name, char
*frame_id_name)
{
 range_name.radiation_type = sensor_msgs::Range::ULTRASOUND;
  range_name.header.frame_id = frame_id_name;
  range_name.field_of_view = 0.26;
  range_name.min_range = 0.0;
  range_name.max_range = 2.0;
}

//Cria instâncias para mensagens de distância.
sensor_msgs::Range range_1;
sensor_msgs::Range range_2;
sensor_msgs::Range range_3;
sensor_msgs::Range range_4;
//sensor_msgs::Range range_5;
//sensor_msgs::Range range_6;
//sensor_msgs::Range range_7;
//sensor_msgs::Range range_8;


//Cria objetos ROS de todos os sensores para publicação
```

```
ros::Publisher pub_range_1("/sonar1", &range_1);
ros::Publisher pub_range_2("/sonar2", &range_2);
ros::Publisher pub_range_3("/sonar3", &range_3);
ros::Publisher pub_range_4("/sonar4", &range_4);
//ros::Publisher pub_range_5("/sonar5", &range_5);
//ros::Publisher pub_range_6("/sonar6", &range_6);
//ros::Publisher pub_range_7("/sonar7", &range_7);
//ros::Publisher pub_range_8("/sonar8", &range_8);

void onTwist(const geometry_msgs::Twist& msg)
{
  if(msg.linear.x > 0)  //TUPY PARA FRENTE
  {
    digitalWrite(IN1,HIGH); //roda direita para frente ON
    digitalWrite(IN2,LOW);  //roda direita para trás OFF
    digitalWrite(IN3,HIGH); //roda esquerda para frente ON
    digitalWrite(IN4,LOW);  //roda esquerda para trás OFF
  }
  else if(msg.linear.x < 0)  //TUPY PARA TRÁS
  {
    digitalWrite(IN1,LOW);  //roda direita para frente OFF
    digitalWrite(IN2,HIGH); //roda direita para trás ON
    digitalWrite(IN3,LOW);  //roda esquerda para frente OFF
    digitalWrite(IN4,HIGH); //roda esquerda para trás ON
  }
  else if(msg.angular.z < 0)   //TUPY GIRA PARA ESQUERDA
  {
    digitalWrite(IN1,HIGH); //roda direita para frente ON
    digitalWrite(IN2,LOW);  //roda direita para trás OFF
    digitalWrite(IN3,LOW);  //roda esquerda para frente OFF
    digitalWrite(IN4,HIGH); //roda esquerda para trás ON
  }
  else if(msg.angular.z > 0)//TUPY GIRA PARA DIREITA
  {
    digitalWrite(IN1,LOW); //roda direita para frente OFF
    digitalWrite(IN2,HIGH);//roda direita para trás ON
    digitalWrite(IN3,HIGH);//roda esquerda para frente ON
    digitalWrite(IN4,LOW); //roda esquerda para trás OFF
  }
  else //PARAR TUPY
  {
    digitalWrite(IN1,LOW); //roda direita para frente OFF
    digitalWrite(IN2,LOW); //roda direita para trás OFF
    digitalWrite(IN3,LOW); //roda esquerda para frente OFF
    digitalWrite(IN4,LOW); //roda esquerda para trás OFF
  }
```

```
}
ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel",onTwist);
//ros::NodeHandle nh;

void setup()
{

  pinMode(IN1,OUTPUT); //Configura a variável IN1 (pin 10)
como saída digital
  pinMode(IN2,OUTPUT); //Configura a variável IN2 (pin 11)
como saída digital
  pinMode(IN3,OUTPUT); //Configura a variável IN3 (pin 12)
como saída digital
  pinMode(IN4,OUTPUT); //Configura a variável IN4 (pin 13)
como saída digital

  nh.initNode();  //inicia o nó ROS para o processo
  nh.subscribe(sub);

  pingTimer[0] = millis() + 75;
  for (uint8_t i = 1; i < SONAR_NUM; i++)
    pingTimer[i] = pingTimer[i - 1] + PING_INTERVAL;

  nh.initNode();
 nh.advertise(pub_range_1);
 nh.advertise(pub_range_2);
 nh.advertise(pub_range_3);
 nh.advertise(pub_range_4);
// nh.advertise(pub_range_5);
// nh.advertise(pub_range_6);
// nh.advertise(pub_range_7);
// nh.advertise(pub_range_8);

  sensor_msg_init(range_1, "/sonar1");
  sensor_msg_init(range_2, "/sonar2");
  sensor_msg_init(range_3, "/sonar3");
  sensor_msg_init(range_4, "/sonar4");
// sensor_msg_init(range_5, "/sonar5");
// sensor_msg_init(range_6, "/sonar6");
// sensor_msg_init(range_7, "/sonar7");
// sensor_msg_init(range_8, "/sonar8");

}

void loop() {
  if (isTimeForLoop(LOOPING)) {
```

```
    sensorCycle();
    oneSensorCycle();
    applyKF();
    range_1.range = oneSensorKalman;
    range_2.range = twoSensorKalman;
    range_3.range = threeSensorKalman;
    range_4.range = fourSensorKalman;
//    range_5.range = fiveSensorKalman;
//    range_6.range = sixSensorKalman;
//    range_7.range = sevenSensorKalman;
//    range_8.range = eightSensorKalman;

    range_1.header.stamp = nh.now();
    range_2.header.stamp = nh.now();
    range_3.header.stamp = nh.now();
    range_4.header.stamp = nh.now();
//    range_5.header.stamp = nh.now();
//    range_6.header.stamp = nh.now();
//    range_7.header.stamp = nh.now();
//    range_8.header.stamp = nh.now();

   pub_range_1.publish(&range_1);
   pub_range_2.publish(&range_2);
   pub_range_3.publish(&range_3);
   pub_range_4.publish(&range_4);
//   pub_range_5.publish(&range_5);
//   pub_range_6.publish(&range_6);
//   pub_range_7.publish(&range_7);
//   pub_range_8.publish(&range_8);

    startTimer();
  }
  nh.spinOnce();//Informa ao ROS que uma nova mensagem chegou.
}
```