```
#include <ros.h> //dedicada à comunicação dos pacotes ROS
#include <ros/time.h> //responsável pela sincronização dos
tempos ROS
#include <sensor_msgs/Range.h> //trata do tipo de mensagem
ROS
#include <NewPing.h> //biblioteca dedicada ao uso de sonares
#include <SimpleKalmanFilter.h> //biblioteca de filtros
digitais

#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 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.

NewPing sonar[SONAR_NUM] = //cria objetos newPing para todos
```

```
os sensores.
{
  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)
};
//criar objetos de filtro Kalman para os sensores.
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);

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

void sensorCycle() //loop em todos os sensores
{
  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]);
}

int returnLastValidRead(uint8_t sensorArray, uint8_t cm)
//Se o valor do sensor for 0, então retorna o último valor
armazenado diferente de 0.
{
  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);
}

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;

//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);

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

  nh.initNode(); //inicia o nó ROS para o processo
 nh.advertise(pub_range_1);
 nh.advertise(pub_range_2);
 nh.advertise(pub_range_3);
 nh.advertise(pub_range_4);

 sensor_msg_init(range_1, "/sonar1");
 sensor_msg_init(range_2, "/sonar2");
 sensor_msg_init(range_3, "/sonar3");
 sensor_msg_init(range_4, "/sonar4");

}

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_1.header.stamp = nh.now();
    range_2.header.stamp = nh.now();
    range_3.header.stamp = nh.now();
    range_4.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);

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