Application of Paraconsistent Annotated Logic in Prototype of Autonomous Vehicle in 1:24 Scale

Henry Costa Ungaro
Graduate Program in Production Engineering
Paulista University
São Paulo, Brazil
henry@paradecision.com

Jair Minoro Abe
Graduate Program in Production Engineering
Paulista University
São Paulo, Brazil
jairabe@uol.com.br

Fábio Vieira do Amaral
Paulista University
São Paulo, Brazil
fabio@paradecision.com

Kazumi Nakamatsu
University of Hyogo
Hyogo, Japan
nakamatu@shse.u-hyogo.ac.jp

***Abstract* — Google, Tesla, and GM are companies that worry about creating a IA-based stand-alone vehicle. These vehicles comprehend the world depends on the data extraction from sensors, radars, cameras, among other devices. One detail that must be considered is the inconsistencies, which appear to be caused by the conditions of the environment in which the evidence is placed. This paper applies the concepts of Paraconsistent Annotated Evidential Et in an embedded software environment from Arduino Uno microcontroller board, ultrasonic sensors, DC motors, vehicle chassis available in Arduino basic kit, in 1:24 scale. The project is to provide an initial knowledge base that can evolve into a more complex situation. The scope of this work is limited to the identification of obstacles and the application of actions that avoid the collision. As proposition: "there are no obstacles ahead". During the tests, the prototype easily recognized obstacles that occur by adopting the measurements determined by the twelve logical states.**

Keywords—paraconsistent logic; paraconsistent annotated logic; autonomous vehicle; Arduino

I. Introduction

Autonomous vehicles tend to benefit society, referring to locomotion, ensuring more safety in critical conditions, reducing the stress generated by large cities' traffic, and others. [1]

II. Theoretical Foundation

Decision making is the cognitive process by which a plan of action is chosen from several others (based on various scenarios, environments, analyzes, and factors) for a problem situation. Every decision-making process produces a final choice. The output may be an action or an opinion. Decision-making refers to the process of choosing the most appropriate path in a given circumstance. [2]

In the real world, we deal with uncertainties, situations of inconsistencies, and often we have only a partial recognition of facts and objects – However, this does not prevent the development of human reasoning that is beyond the binary relation of truth and falsity [3]. The need to demonstrate and treat contradictory and non-trivial situations led to the emergence of an underlying logic for formal systems called paraconsistent logics [4].

A. Paraconsistent logic

The necessity to make decisions occurs at a moment of deadlock, which there are more than one option to follow. We make decisions based on subjective aspects; subjectivity has no perfect measure; it is organized, systematically and objectively. [2]

Paraconsistent Logic is among the non-classical logical since it contains provisions contrary to some of the basic principles of Aristotelian Logic, such as the principle of contradiction. Under Aristotelian view, any statement is necessarily true or false. According to the Paraconsistent Logic, a sentence and its negation may both be true [4]. It works with propositions of type p (μ, λ), where p is a proposition and (μ, λ) indicate the degrees of favorable evidence and contrary evidence, respectively. The pair (μ, λ) is called the annotation constant, with the values of μ and λ being limited between 0 and 1 [5]. The input data processing takes place through the application of minimization and maximization connectives between the atomic formulas A and B that define the output state, considering the propositional ones with their respective degrees of certainty and uncertainty pA (μ_1, λ_1) and pB (μ_2, λ_2), the highest value is obtained between the degrees of certainty (μ_1 OR μ_2), obtaining the resulting degree of certainty (μ_R), then minimizing the degrees of uncertainty (λ_1 OR λ_2) obtaining the degree of resulting uncertainty (λ_R) [5].

Considering the scenario of two expert groups A (E1, E2) and B (E3, E4), we can demonstrate the application of the OR connective represented by the disjunction A v B:

E1 (μ_1, λ_1) OR E2 (μ_2, λ_2) = (Max $\{\mu_1, \mu_2\}$, Min $\{\lambda_1, \lambda_2\}$) = AR (μ_1, λ_1)

E3(μ_1, λ_1) OR E4(μ_2, λ_2) = (Max $\{\mu_1, \mu_2\}$, Min $\{\lambda_1, \lambda_2\}$) = AR (μ_2, λ_2)

Then the application of the AND connective between the annotated AR and BR signals, representing the AR Conjunction $\wedge$ BR:

R = AR (μ_1, λ_1) AND BR (μ_2, λ_2) = (Min $\{\mu_1, \mu_2\}$, Max $\{\lambda_1, \lambda_2\}$) = R (μ_1, λ_1)

After maximization and minimization, the degrees of certainty and uncertainty are obtained by:

• Degree of certainty: Gce(μ, λ) = μ - λ
• Degree of Uncertainty: Gun(μ, λ) = μ + λ - 1

Two external and arbitrary boundary values (Vcve = Truth control value and Vcfa = False control value) determine when the resulting degree of certainty is high enough that the proposition analyzed is considered totally true or totally false.

Likewise, two external and arbitrary boundary values (Vcic = Control value of inconsistency and Vcpa = Control value of paracompleteness) determine when the value of the degree of uncertainty resulting from the analysis is so high that the proposition can be considered totally inconsistent or totally paracomplete (Table 1).

TABLE I. EXTREME VALUES [6]

External Limit Values	
Vcve	Truth control value
Vcfa	False control value
Vcic	Inconsistency control value
Vcpa	Paracomplete control value

After determining the four limit values and the results of the degree of certainty and uncertainty, it is possible to identify the resulting logical state. Through the use of such concepts, we arrive in Figure 1.

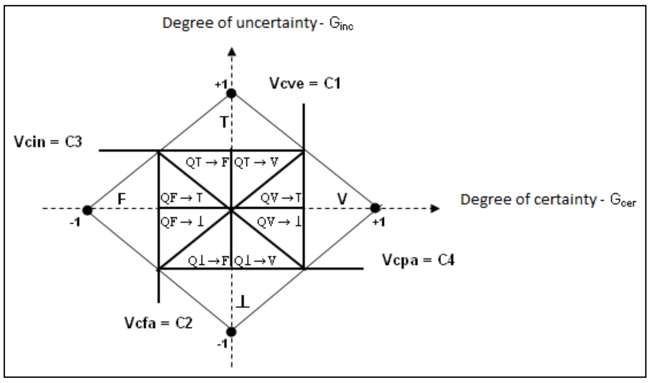

Fig. 1. Diagram with the degrees of certainty and uncertainty, with adjustable values of limit control, indicated in the axes [6]

The logical states which are represented by regions that occupy the vertices of the lattice are: True, False, Inconsistent and Paracomplete. These are called extreme logic states. The output logic states represented by internal regions in the lattice that is not the extreme logic states are called non-extreme logic states. Each non-extreme logical state is named according to its proximity to the extreme logic states.

The following are four logical states extreme Table 2 and eight non-extreme Table 3 that make up the lattice of Figure 2.

TABLE II. EXTREME STATES [6]

Extreme State	Symbol
True	V
False	F
Inconsistent	T
Paracomplete	⊥

TABLE III. NON-EXTREME STATES [6]

Non-Extreme State	Symbol
Quasi-true tending to Inconsistent	QV→T
Quasi-true tending to Paracomplete	QV→⊥
Quasi-false tending to Inconsistent	QF→T
Quasi-false tending to Paracomplete	QF→⊥
Quasi-Inconsistent tending to True	QT→t
Quasi-Inconsistente tending to False	QT→F
Quasi-Paracomplete tending to True	Q⊥→V
Quasi-Paracomplete tending to False	Q⊥→F

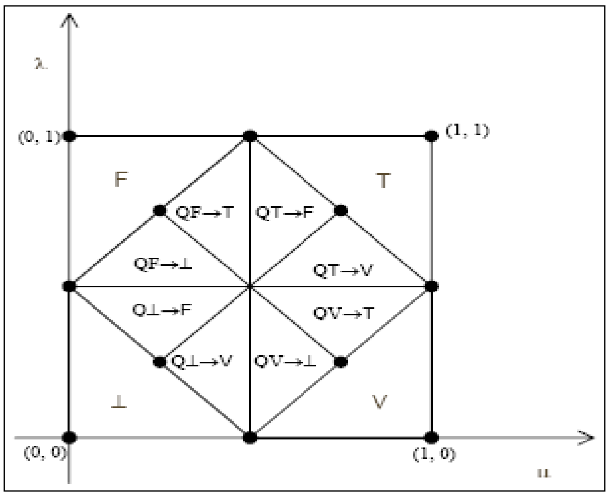

Fig. 2. Division of the lattice in 12 regions [6]

The characterization the resulting logical states, the following rules are considered (Table 4):

TABLE IV. MATHEMATICAL CHARACTERIZATION OF THE STATES [5]

Condition	Resulting State
If Gcer(μ, λ) ≥ Vcve	True
If Gcer(μ, λ) ≤ Vcfa	False
If Ginc(μ, λ) ≥ Vcic	Inconsistent
If Ginc(μ, λ) ≤ Vcpa	Paracomplete
If 0 ≤ Gcer(μ, λ) < Vcve and 0 ≤ Ginc(μ, λ) < Vcic and Gcer(μ, λ) ≥ Ginc(μ, λ)	Quasi-true tending to Inconsistent
If 0 ≤ Gcer(μ, λ) < Vcve and 0 ≤ Ginc(μ, λ) < Vcic and Gcer(μ, λ) < Ginc(μ, λ)	Quasi-Inconsistent tending to true
If 0 ≤ Gcer(μ, λ) < Vcve and Vcpa < Ginc(μ, λ) ≤ 0 and Gcer(μ, λ) ≥ |Ginc(μ, λ)|	Quasi-true tending to Paracomplete
If 0 ≤ Gcer(μ, λ) < Vcve and Vcpa < Ginc(μ, λ) ≤ 0 and Gcer(μ, λ) < |Ginc(μ, λ)|	Quasi-Paracomplete tending to true
If Vcfa < Gcer(μ, λ) ≤ 0 and Vcpa < Ginc(μ, λ) ≤ 0 and |Gcer(μ, λ)| ≥ |Ginc(μ, λ)|	Quasi-false tending to Paracomplete
If Vcfa < Gcer(μ, λ) ≤ 0 and Vcpa < Ginc(μ, λ) ≤ 0 and |Gcer(μ, λ)| < |Ginc(μ, λ)|	Quasi-Paracomplete tendending to False
If Vcfa < Gcer(μ, λ) ≤ 0 and 0 ≤ Ginc(μ, λ) < Vcic and |Gcer(μ, λ)| ≥ Ginc(μ, λ)	Quasi-false tending to Inconsistent'

If $Vcfa < Gcer(\mu, \lambda) \leq 0$ and $0 \leq Ginc(\mu, \lambda) < Vcic$ and $\lvert Gcer(\mu, \lambda)\rvert < Ginc(\mu, \lambda)$	Quasi-inconsistent tending to False

B. *Hardware*

Arduino is an open source hardware platform, designed on the Atmel AVR microcontroller, which can be programmed through a programming language similar to C / C ++, allowing the preparation of projects with a basic or no programming and electronic knowledge. [7]

Motors and H-Bridge. The basic principle of DC motors is to let the electric current flow through a coil, creating a magnetic field. This magnetic field applied to a magnet results in the rotation of the shaft, which may be connected to wheels, propellers or any other type of gear. [7]

The H-Bridge is an integrated circuit that facilitates the assembly of circuits for the use of motors, allowing the movement of these motors clockwise and counter clockwise. These plates protect the motor circuit of the others, avoiding damages. [9]

Ultrasonic Sensor. The ultrasonic sensor HC-SR04 allows detecting objects that are in the distance between 1 and 200 cm.

This sensor emits an ultrasonic signal that reflects in an object and returns to the sensor, allowing to calculate the distance of the object concerning the sensor, adopting as a base the time of trajectory of the signal. [7]

Chassis. The chosen chassis was the standard model of the kits supplied with the Arduino microcontroller. Acrylic structure, with three wheels being two associated with motors and the third wheel, formed by bearing without motor control. [10]

C. *Methodology*

Experimental implementation of paraconsistent logic concepts through the construction of a prototype based on the Arduino platform

III. PROTOTYPE

Figure 4 shows the circuit with all the components used.

PowerBank Lotus LT55, lithium battery with a capacity of 10000mAh @ 3.7V, DC input 5V 2A output DC 5V 1A / 2.1A output:> 6800MAH> 31.5WH, with two USB inputs where the USB1 feeds Arduino and USB2 power the motors.

Two ultrasonic sensors were used, in which one corresponded to a "favorable degree of evidence" and the other to "opposite degree of evidence." Arduino pins 4, 5, 6 and seven are used to control the two motors connected to H-Bridge.

The pins 9 (Trigger) and 12 (Echo) is responsible for controlling the left-hand ultrasonic (μ) and the pins 10 (Trigger) and 13 (Echo) the right (λ) pins.

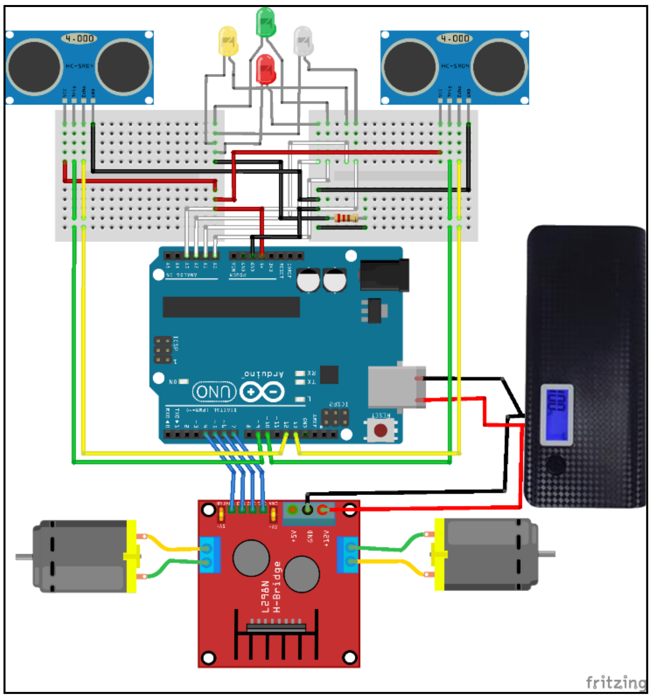

Fig. 3. Prototype Wiring Scheme

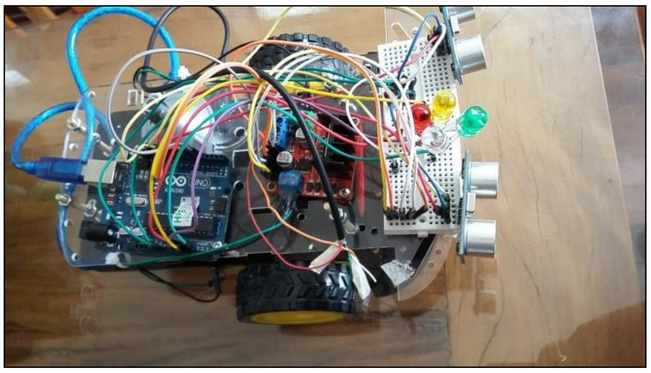

Fig. 4. Prototype

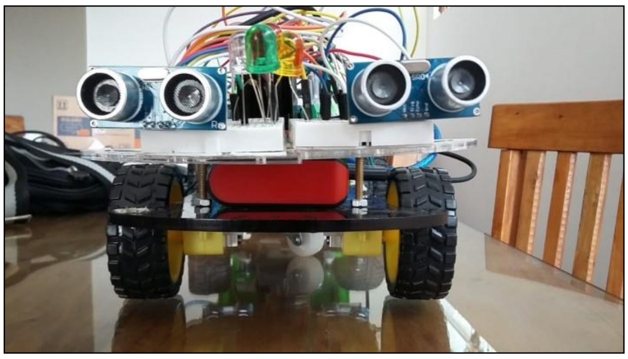

Fig. 5. Prototype

IV. EVENT DEFINITIONS

As proposition, it was considered that there are no obstacles in front of the vehicle.

Maximum distance was taken by sensors: 120 cm.

For maximum distance, was assigned μ value 1 and for λ value 0, in correspondence for the minimum distance, was assigned μ value 0 and for λ value 1. For control values, Vcve was assigned +0 value, 5, for the Vcfa was assigned value -0.5, for the Vcic was assigned +0.5 value and for Vcpa was assigned value -0.5. Figures 11 and 12 correlate extreme and non-extreme logic states with regions that were considered as possible obstacle holders. The center line comprises the perfectly defined line, where the degree of certainty becomes more decisive about the presence of obstacles. As it moves away from the center line towards the vertical extremes, the level of inconsistency and indetermination increases, as a consequence, the actions referring to the states near the center line and μ tending to 0 indicate the presence of an obstacle closer and closer to the vehicle. Therefore, more actions should be taken.

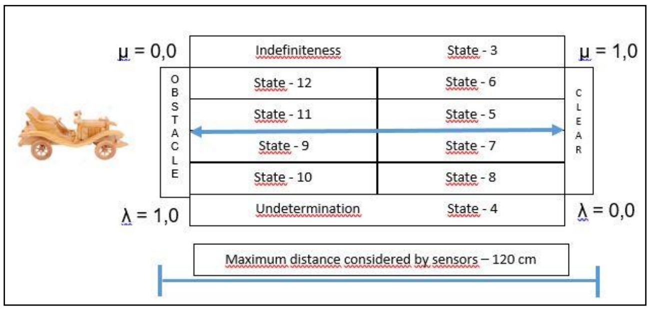

Fig. 6. Prototype decisions in logic state

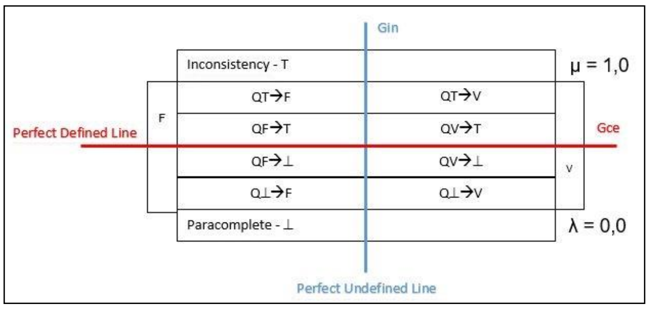

Fig. 7. Prototype extreme state

V. SOURCE CODE

```
#include <Ultrasonic.h>
//Ultrasonic pins
#define pino_trigger_mi     9       // The sensor sends a
     ultrasonic wave
#define pino_trigger_lambda 10      // The object reflect this
     wave and
#define pino_echo_mi        12      // Echo recive the wave
#define pino_echo_lambda    13
//Ultrasonic Start Up
Ultrasonic sensor_mi(pino_trigger_mi, pino_echo_mi);
Ultrasonic sensor_lambda(pino_trigger_lambda,
     pino_echo_lambda);
// Control Variables
float distancia_mi;                // distance value for sensor_mi
float distancia_lambda;            // distance value for
     sensor_lambda
float vcve = 0.5;                  // control variable for true
float vcfa = -0.5;                 // control variable for false
float vcic = 0.5;                  // control variable for
     inconsistency
float vcpa = -0.5;                 // control variable for de
     paracomplete
```

```
// ParaAnaliser
int paraAnalisador(float mi, float lambda) {
  // Normalization of evidence degree between 0 and 1
  mi   = mi / 100;                     // Favorable degree -  0 , 1
  lambda = lambda / 100;               // Unfavorable degree -
     0 , 1
  float Gce  = mi - lambda;            // Gce - certainty
     degree  - Gce = mi - lambda
  float Gin  = ((mi + lambda) - 1);    // Gin - uncertainty
     degree - Gin = mi + lambda - 1
  int estado = 0;                      // Logic States, Extreme and
     Non-Extreme
  float modulo_Gce;                    // Module Value for
     certainty
  float modulo_Gin;                    // Module Value for
     uncertainty
  if (Gce < 0)
    modulo_Gce = Gce * (-1);
  else
    modulo_Gce = Gce;
  if (Gin < 0)
    modulo_Gin = Gin * (-1);
  else
    modulo_Gin = Gin;
  // Extreme states definition
  // Proposition: path ahead is clear
  if(Gce >= vcve)
    estado = 1; //true - path is clear
  else if(Gce <= vcfa)
        estado = 2; //False - it will hit - Stop, backwards, turn
     right and left
       else if(Gin >= vcic)
            estado = 3; //Inconsistent - turn slightly right
           else if(Gin <= vcpa)
                estado = 4; //Paracompleto - turn slightly left
               else if( (Gce >= 0) && (Gce < vcve) && (Gin >=
     0) && (Gin < vcic) && (Gce >= Gin))
                    estado = 5; //Quasi-true tending to
     inconsistent - Turn right, more than state 3
                   else if((Gce >= 0) && (Gce < vcve) && (Gin
     >= 0) && (Gin < vcic) && (Gce < Gin))
                        estado = 6; //inconsistent tending to true -
     turn slightly left , less than state 5
                       else if((Gce >= 0) && (Gce < vcve) &&
     (Gin > vcpa) && (Gin <= 0) && (Gce >= modulo_Gin))
                            estado = 7; //Quasi-true tending
     paracomplete- turn left, more than state 8
                           else if((Gce >= 0) && (Gce < vcve) &&
     (Gin > vcpa) && (Gin <= 0) && (Gce < modulo_Gin))
                                estado = 8; // paracomplete tending
     to true - turn slightly left , more than state 4
                               else if((Gce > vcfa) && (Gce <= 0)
     && (Gin > vcpa) && (Gin <= 0) && (modulo_Gce >=
     modulo_Gin))
                                    estado = 9; // quasi-false tending
     to paraconsistent - Stop, turn left
                                   else if((Gce > vcfa ) && (Gce <=
     0) && (Gin > vcpa) && (Gce < Gin) && (Gin <= 0))
                                        estado = 10; // paracomplete
     tending to false - Stop, turn left
```

```
                                else if((Gce > vcfa) && (Gce
     <= 0) && (Gin >= 0) && ( Gin < vcic) && (Gce >=
     Gin))
                                    estado = 11; // quasi-false
     tending to inconsistent- Stop, turn right
                                   else if((Gce <= 0) && (Gce
     < vcfa) && (Gin >= 0) && (Gin < vcic) && (Gce <
     Gin))
                                       estado = 12;
     //inconsistent tending to false - Stop, turn slightly right
  return estado;
}
// H-Bridge variables (L293D)
int in1 = 7;            // input 1
int in2 = 6;            // input 2
int in3 = 5;            // input 3
int in4 = 4;            // input 4
// Distance ajustment
float ajusteDistancia(Ultrasonic sensor) {
  float cmMsec;
  long microsec = sensor.timing();
  cmMsec = sensor.convert(microsec, Ultrasonic::CM);
  if (cmMsec > 120)           //Define maximum distance
    cmMsec = 120;
  else if (cmMsec < 5)        //Define minimum distance
    cmMsec = 5;
  return cmMsec;
}
void setup() {
  Serial.begin(9600);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(verde_verdadeiro     , OUTPUT);
  pinMode(vermelho_falsidade   , OUTPUT);
  pinMode(amarelo_inconsistente, OUTPUT);
  pinMode(branco_paracompleto  , OUTPUT);
}
// Motor Control
void para(){
    digitalWrite(in2,LOW);
    digitalWrite(in1,LOW);
    digitalWrite(in3,LOW);
    digitalWrite(in4,LOW);
}
void anda(){
    digitalWrite(in1,LOW);
    digitalWrite(in3,HIGH);
    digitalWrite(in2,HIGH);
    digitalWrite(in4,LOW);
}
void re(){
    digitalWrite(in1,HIGH);
    digitalWrite(in3,LOW);
    digitalWrite(in2,LOW);
    digitalWrite(in4,HIGH);
}
void direita(){
    digitalWrite(in1,LOW);
    digitalWrite(in3,LOW);
    digitalWrite(in2,HIGH);
    digitalWrite(in4,LOW);
}
void esquerda(){
    digitalWrite(in1,LOW);
    digitalWrite(in3,HIGH);
    digitalWrite(in2,LOW);
    digitalWrite(in4,LOW);
}
void esquerda_f(){
    digitalWrite(in1,HIGH);
    digitalWrite(in3,HIGH);
    digitalWrite(in2,LOW);
    digitalWrite(in4,LOW);
}
void direita_f(){
    digitalWrite(in1,LOW);
    digitalWrite(in3,LOW);
    digitalWrite(in2,HIGH);
    digitalWrite(in4,HIGH);
}
// --- LOOP ---
void loop() {
  distancia_mi = map(ajusteDistancia(sensor_mi), 10, 120,
     0, 100);
  distancia_lambda = map(ajusteDistancia(sensor_lambda),
     10, 120, 100, 0);
  int estado =
     paraAnalisador(distancia_mi,distancia_lambda);
  Serial.println(String("Distance-mi : ") + distancia_mi +
     String("| Distance-lambda : ") + distancia_lambda +
     String("| State : ") + estado );
  if(estado == 1){
    anda();
  }
  else if(estado == 2){
    re();
  }
  else if(estado == 3){
    direita_f();
    anda();
    esquerda_f();
  }
  else if(estado == 4){
    esquerda_f();
    anda();
    direita_f();
  }
  else if(estado == 5){
    direita_f();
  }
  else if(estado == 6){
    direita();
  }
  else if(estado == 7){
    esquerda_f();
  }
  else if(estado == 8){
    esquerda();
  }
  else if(estado == 9){
    para();
    esquerda_f();
  delay(500);
 }
 else if(estado == 10){
  para();
  esquerda();
 }
 else if(estado == 11){
  para();
  direita_f();
  delay(500);
 }
 else if(estado == 12){
  para();
  direita();
 }
}
```

VI. Conclusions

During the tests, all the logical states were identified, when facing obstacles, in diagonal, the position of the sensors did not prove useful and are in need of adjustments. Although the hardware limitations, the decision making process proved to be efficient in relation of response time, deviating obstacles with relative ease, the number of collisions presented an index with less than 5% in relation of sample universe formed by 123 obstacles.

References

[1] Jung, C.R. et al. Computação embarcada: Projeto e implementação de veículos autônomos inteligentes. Anais do CSBC (in Portuguese), v. 5, p. 1358-1406, 2005.

[2] Shimizu, T. (2006). Decisão nas Organizações (Vol. 2 ed.) (in Portuguese), São Paulo, SP, Brasil: Atlas, 2006.

[3] Martins, H. G. (2003). A Lógica Paraconsistente Anotada de Quatro Valores LPA4v aplicada em Sistemas de Raciocínio Baseado em Casos para o Restabelecimento de Subestações Elétricas. Tese de Doutorado apresentada à Universidade Federal de Itajubá, 2003.

[4] Akama, S., “Towards Paraconsistent Engineering,” Intelligent Systems Reference Library, Volume 110, 234 pages, 2016.

[5] Abe, J.M., S. Akama, K. Nakamatsu, “Introduction to Annotated Logics - Foundations for Paracomplete and Paraconsistent Reasoning,” Series Title Intelligent Systems Reference Library, Volume 88, Springer International Publishing, Copyright Holder Springer International Publishing Switzerland, ISBN 978-3-319-17911-7, Edition Number 1, 190 pages, 2015.

[6] Abe, J.M., “Paraconsistent Intelligent Based-Systems: New Trends in the Applications of Paraconsistency,” editor, Book Series: “Intelligent Systems Reference Library,” Springer-Verlag, Vol. 94, ISBN:978-3-319-19721-0, 306 pages, 2015.

[7] Oliveira, C.L.V., H.A.P. Zanetti, Descomplicado: como elaborar projetos de eletrônica (in Portuguese), Saraiva Educação SA, 2015.

[8] Fritzing. Available at: http://fritzing.org/home/ Retrieved 2018-11-10

[9] Thomsen, Adilson. Motor DC com Driver Ponte H L298N. 2013. Available in: https://www.filipeflop.com/blog/motor-dc-arduino-ponte-h-l298n/. Retrieved 2018-11-10

[10] MSS Eletrônica Available at: https://www.msseletronica.com/detalhes/kit-chassis-2-rodas-2-motores-eletricos-dc-cc-para-robos-robotica-carrinho-2wd/990.html. Retrieved 10/11/2018