ITAndroids Very Small Size Soccer Team Description Paper for LARC 2020*

Kenji de Souza Yamane[1], Nicholas Scharan Cysne[1], Levy Bruno Batista[1], Thiago Filipe de Medeiros[1], Guilheme Silva de Oliveira[1], Eduardo Santos Guimarães[1], Thayná Pires Baldão[1] and Marcos R. Omena Máximo[2]

***Abstract*— ITAndroids Very Small Size League group was created in the middle of 2013 by undergraduate students at Technological Institute of Aeronautics (ITA). Our objective at that time was to participate of the Brazilian Robotics Competition (CBR), though unfortunately we did not have enough time to build a functional team (we only had made the robots, we had no software). The following years were dedicated to create the whole vision system and the strategy of the game. Last year (2019) we participated of CBR and had very good results, third place in the 3v3 category and first place in 5v5. In this team description report, we want to detail what we did, both in hardware of the robots and the computer software.**

I. INTRODUCTION

Since 2012, some members of ITAndroids, a robotics team from Technological Institute of Aeronautics (ITA), wished to participate of a non-simulated robot soccer competition, but we had no knowledge about it.

Thus, it was clear that it would be better for the students to take part in a low-cost project in which we could learn a lot about hardware, vision processing, control and strategy algorithms. Very Small Size League meets exactly these requirements. This paper describes the evolution of the team and detail the most relevant parts of the project.

[1]University graduate student at Technological Institute of Aeronautics, São José dos Campos, São Paulo, Brazil

[2]Doctor at Technological Institute of Aeronautics, São José dos Campos, São Paulo, Brazil `maximo.marcos@gmail.com`

II. THE HARDWARE

A. The Electronics

First, we will expose the electronics. It is very simple and is composed basically of a microcontroller, an H bridge, encoders for each wheel, motors and a radio.

1) The Radio Module: The nRF24L01 radio module, built by Nordic Semiconductor, carries all the communication between que computer and each robot. Each radio has one TX pipe, allowing it to send messages to one radio at time, but 6 RX pipes, making possible to the computer receive messages from all 5 robots at the same time.

This radio works with a carrier frequency of 2.4 GHz and it is able to send messages with data rate of 2 Mbps. It is worth noting that the message size is limited: a maximum of 32 bytes can be sent from one module to another.

VSS team has five robots and one computer, so 6 radios are needed. Since this radio uses the SPI communication interface, the team needed to develop a USB dongle using an Arduino UNO to allow communication with the radio and the computer. Almost all the radio functions are implemented in the ITAndroids-VSS GitLab C++ code. So, after the Arduino Dongle firmware is uploaded, it is possible to control almost all the radio using only the CLion IDE, building code only in the computer.

An open source driver library was employed to handle all the required steps (internal radio registers configuration) to one radio send or receive a message. The employed original library is available at `https://github.com/nRF24/RF24`.

Since this library is originally written to Arduino, it was rewritten to work with the STM CubeHAL of the STM32F303K8 microcontroller.

2) The Motors, Encoders and Motor Driver: Our robot needs two motors (1 for each wheel). These actuators are the main responsibles for the translational and rotational movements of the robot.

The motors we used for this project were 2224R 006SR + IE2 512 (rising edge pulses per channel). These motors have 6V nominal voltage, with a 2048 pulses per turn magnetic encoder. The motor connector it's an IDC (Insulation-displacement) connector. The main advantages of this motor are the high precision encoders, and the excellent performance for a brush DC motor.

The microcontroller can't feed the actuators alone, so it needs another subsystem to drive the motors. This subsystem is called Motor Driver (or H Bridge). Our current motor driver is DRV8872-Q1 (DRV8872-Q1 Automotive 3.6-A Brushed DC Motor Driver With Fault Reporting). Reasons to be chosen: supports motor current, works well with high PWM frequency and the IC is very small.

This motor driver has two operating ways with respect to current circulation inside the H Bridge: fast decay and slow decay mode. The description of the 2 modes can be found with further information in the component's datasheet. Since our motor has low inductance, and operate with a maximum 3.0A stall current, we chose to use the slow decay motor driver mode over the fast decay mode.

3) The Microcontroller: In 2018, the Nucleo F303K8 was replaced by the microcontroller itself, STM32F303RET6K8, by space saving. To this, it was necessary to add a clock and a reset circuit. Because of the clock circuit, it is possible to use the microcontroller at the clock frequency of 72 MHz. Since we are using the microcontroller itself, it was necessary to use an programmer device. We used the ST-LINK/V2.

4) The Power Supply: We chose a 7.4 V LiPo battery to supply power to the whole circuit. LiPo's are good for their high discharge rate, high charge capacity and light weight. We could connect the battery directly to the motors and to Nucleo. The NRF24L01 requires 3.3V with 5 V, both obtained from Nucleo regulators.

5) Connecting everything in the PCB: The circuit board was an improvement from last year's version. In this version, the Altium version 18.1 was employed to design the board. A structured design was made and is shown in Fig. 1.

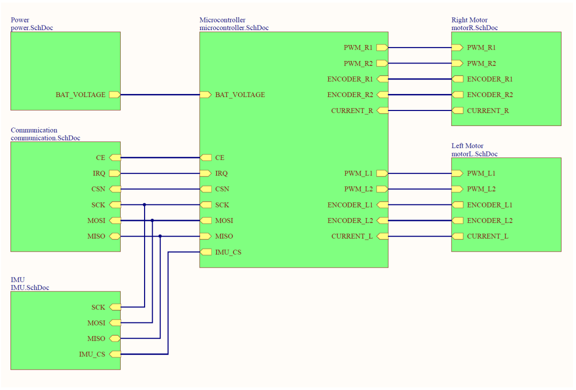

Fig. 1: Top level circuit diagram.

As can be seen from Fig. 1, the microcontroller (central block) is connected to another five blocks: power, communication, IMU and left and right motors. It is worth noting that, in this design, any change in the circuit reflects only in the respectively internal block. For example: when replacing the microcontroller, it is necessary to work only in the microcontroller block.

A resistive divisor was employed to allow the microcontroller to sense the battery voltage. Also two LEDs were added: D2 (red) to indicate that the battery has discharged and D3 (green) to inform that the system is power-up. In addition to this the dip switch SW2 is employed to select the ID of the robot, beginning in 0 (zero) and ending in 4 (four), folowing the binary notation to make this selection.

Besides that a small capacitor was placed very close to the nRF24L01 to avoid malfunctioning of the radio. The IRQ pin is connected to a microcontroller interrupt pin.

The motor maximum current was limited to 2.9 A. Since the motor encoder is 5 V, two resistor divider were employed to convert the 5 V to 3.3 V of the microcontroller.

In addition, by-pass capacitors were required: without them the H-bridge doesn't work. Also, the connector P7 was employed to select the ID of the robot, it is necessary to use old motherboard jumpers to make the selection.

B. The Mechanics

The project can be divided in 4 main parts: the motors block, the reduction block, the circuit block and the cover. An overview of the project is show in Fig. 2.

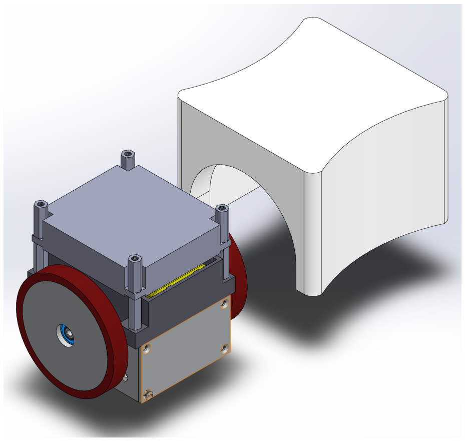

Fig. 2: Overview of the new mechanics.

The reduction block is composed by 9 parts, including 1 shaft, 2 wheels, 2 gears of 119 teeth, 2 bearings of 11mm x 5mm x 5mm and 2 retaining rings of 5mm. The circuit block is composed by one part that is 3D printed. The reason for this is: if it is necessary to redo the PCB, simply modify this part.

The wheel of the robot is made of aluminum, similar to the other parts. The tire was made of Poly 74-20, a Polyurethane Rubber. To manufacture them, some 3D printed molds were used.

C. The Control

The Control Law uses a PI Control Law. Although, our project uses an anti-slippage algorithm, controlling the commanded speed in order to avoid high accelerations (that can cause slippage).

Additionally to the PID control algorithm, it's included in the code lines a procedure that limits the motor's angular acceleration. The procedure takes the reference angular speed, the current angular speed and estimate the necessary acceleration to achieve the reference in one Time step. If the needed acceleration outplay the maximum acceleration allowed by the control designer, the speed commanded by the controller will be calculated using the current motor speed plus the linear increment of the maximum allowed acceleration in one time step. This procedure is a must take, since a high acceleration could make the wheel skid on the drive, disturbing other high level controllers.

D. The Embedded Software

The function of the microcontroller is to process a speed message sent from a computer through a radio and control the speed of both wheels. So the first thing we need is a protocol, that is, we need to create a language that both the computer and the microcontroller understands.

1) Communication Protocol: With the five players per team category, the communication protocol was extended to be possible a five players team, but at the same time made sure that the code would be used for the three players team as well.

The communication protocol is made to make available a bilateral communication between the computer and the robots. Hence each radio has 6 RX pipes, we were able to configure the radio to receive health status messages from each robot every time a message is sent.

2) Robot's Response: Every time a message reaches a robot, a response is immediately fired, containing up-to-date data from the robot's battery level, inner temperature, left and right motor's current. The computer then receives each robot heath status and shows in the GUI these values.

III. The Vision System

The camera used is a Point Grey Flea3 (FL3-U3-13S2C-CS), which runs at 60 Hz using the pixel format YUV422, i.e., 16 bits are used per pixel on average. The current frame resolution is 656 x 516.

The vision processing algorithm can basically be divided in 2 pieces (after calibrating the necessary parameters, of course): a *Blob Detector* and the *Pattern Detection*.

A. Blob Detection

The blob detection is responsible for grouping adjacent pixels that have the same color label into a single object (blob). The first step is to segment the whole image, i.e., we label every pixel with a color and create an image of color labels. Then, we iterate over each image line and group pixels with the same labels into run lengths. Next, we merge run lengths that overlap into blobs, using an union find algorithm with path compression.

After the blobs are identified, we take measures like:

1) Area: is the quantity of pixels inside a blob.
2) Center position: computed using the mean in the x and y position of each blob pixel.
3) Direction: measured using the position matrix covariance. Then, by diagonalizing this matrix, it is possible to find the direction of maximum and minimum covariance, which can be used to estimate the blob direction or the robot orientation.

B. Pattern Detection

The pattern used for position and orientation estimation is shown in Fig. 3. The position estimate of the robot is simply the center of the larger marker, which is determined like it is described in Sec. III-A. The orientation, on the other hand, is found in the following way: after the larger marker is detected, the other two secondary markers are searched within a certain radius and with their positions, vectors corresponding to the difference between the main marker and the secondary ones are determined and then summed.

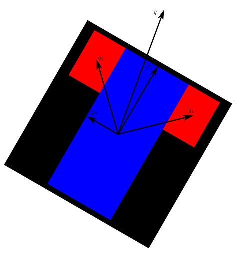

Fig. 3: Robot pattern. It's also illustrated some vectors used for orientation estimation.

IV. The Game Strategy

We adopted a structure to the game strategy that consists of a Finite State Machine which alternates between Coaches, i.e Behaviour Trees [2]. This structure was chosen because of its modularity and relative practicality in changing roles among players. As it was shown in practice, this structure is better than a pure state machine for developing the high level of the strategy, even though it gets more complicated with the lowest level details.

The behaviour tree is composed by nodes that are not leafs - which can be a selector, a sequential, or parallel behaviour or a decorator - and leafs. The first elements are used to choose the action, and the second, play the behaviour.

A. Behaviour tree structure

1) Selector Behaviour: In this node, we select one of its sons to access next. If one of its sons return TRUE or RUNNING, the selector behaviour returns the same value. If one soon returns FALSE, it tries to play the next soon. As soon as it returns TRUE it is not expected for this node to try play any of the remaining sons.

2) Sequential Behaviour: The sequential behaviour tries to execute a queue of actions. It runs each of its sons in order and goes to the next soon only when the one that is running returns TRUE. If any soon returns FALSE, then the sequence behaviour aborts itself.

3) Parallel Behaviour: This behaviour aims to run all its sons. If one of them returns FALSE it also returns FALSE. Otherwise it just keep running the behaviours.

4) Decorator: The decorator has only one son and it changes the state its son returns depending on what the programmer defines. The TrueDecorator, for instance, always returns TRUE.

B. Behaviour Tree VSS implementation

For using the behaviour tree in the ITAndroids VSS code, there was first made a division of the strategy in levels. The lowest level is the behaviour itself, and just above that is the behaviour of one single player. Above that we have a coach, which chooses what player behaviour goes to each player in the game. But we can also have many coaches: one for aggressive defending, other for having a better attack, etc.

C. *Expansion of strategy for VSSS 5x5*

Basically, for 5x5, the implemented strategy follows the same 3x3 strategy-level structuring described, what changes is the complexity of the tasks performed by the coaches and the amount of coaches that can be created, since there are many more robots to assign roles. This year, many new roles were created for the 5x5, such as the second goalier and an intermediate.

D. *Robots positioning*

In addition to all this decision-making, we've also implemented an algorithm to aid in the positioning of assistant robots using *Delaunay triangulation* [3]. Basically, the best positions for the assistants in relation for some positions of the ball on the field are used to map this field with *Delaunay triangulation*. The unmapped points are determined through an interpolation of the vertices of the triangles of the triangulation, using the *Gouraud Shading* algorithm [4]. A GUI was also created to aid in the positioning, showed in Figure 4.

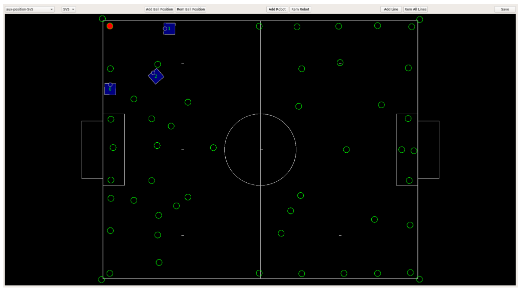

Fig. 4: GUI used to indicate ideal positioning.

E. *Path planning*

After the best destination is determined through the strategy, the path used to get there is also needed to be constructed, which is done using a unitary vector field, where only the desired angle is controlled. This field is constructed by setting attractive ”forces“ relative to the desired destination and repulsive ”forces“ relative to the obstacles in the way. It generates a field such as the one showed in Figure 5, where a path can be determined.

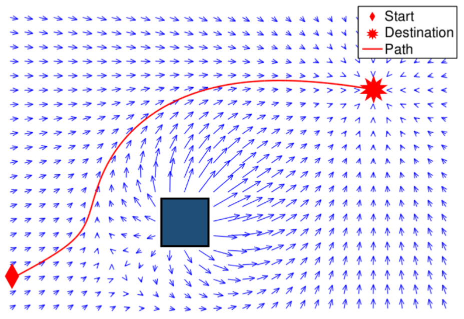

Fig. 5: Path determined through the univector field.

V. Simulator

After adapting the simulator code to sustain 5 robots per team, involving changes such as trading the original judge to a virtual judge from which derives two judges, one for 3x3 and one for 5x5, the statistics were deleloped as well, storing in a text file the following data:

- Number of free balls that happenend on the match;
- Number of goals for each team;
- Number of attack faults committed by each team;
- Number of penalties committed by each team;

Such functional simulator is represented in Figure 6.

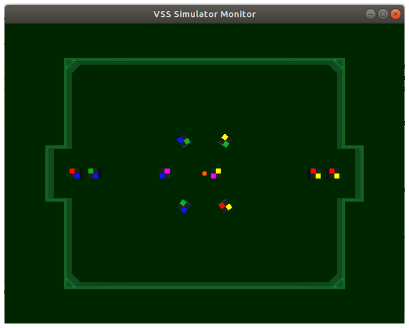

Fig. 6: ITAndroids 5x5 VSSS simulator.

Our simulator uses the model of our robots previously made and, with that, it simulates the strategy of the game using physics, graphics and rules made by our team assisted by tools like Ogre, to render the players and ball, blender, to render the field and ODE to simulate the physics. It also shares the Grafical User Interface used in the main executable. To simulate the vision, the simulator uses a fake webcam and it even simulates the camera delay.

A. Physics

The physics is the part that models the robot and the interactions between different objects. It is built using ODE. It is necessary to include parts of the robot like the wheels, the chassis and even the hinges in this modelling. In the last year, however, the VSSS ITAndroids team developed a new mechanics and electronics for the robots, because of that many parts of the code needed to be adapted. It is also necessary to put the mass of each of those components to help to simulate the collisions, which are the current biggest challenges to the simulator when more than two robots are involved, because it takes too much processing time to calculate and render what happens during the colision, generating latency. The fact that the simulation needs accurate data proves the importance of having a good modelation of the robot.

B. Judge

The judge is the aplication of the rules found at CBR's website [1]. The judge is responsible to detect fouls, like penalties and goal kicks, and stalemates, like when the ball gets stuck in the corner of the field. After that occurs, the robots and the ball are repositioned to the acording places. The judge also keeps the score and the time of the game with him. A thing to be considered is that some rules are open to interpretation and, in the computer, we need exact things, so the judge is very severe about the rules, what is a bit different than the real game.

C. Communication

The communication on the simulator uses a Google library, Protubuff, to parse mensages and communicate with the simulated environment. The siulator communication protocol is shown in Fig. 7.

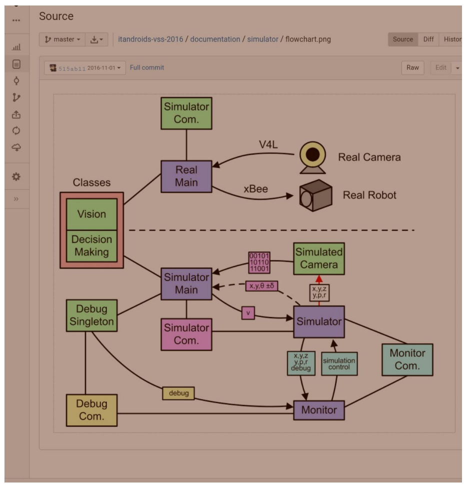

Fig. 7: Simulator Communication Protocol.

ACKNOWLEDGMENT

We thanks this work to Technological Institute of Aeronautics, Altium, Cenic, Intel, ITAEx, Mathworks, Metinjo, Micropress, Polimold, Rapid, SolidWorks, STMicroeletronics, Wildlife Studios and Virtual.PYXIS.

REFERENCES

[1] IEEE. VSS Rules. www.cbrobotica.org [Online; accessed Jan 16th, 2018]. 2008.
[2] COLLEDANCHISE, Michele. Behavior Tree In Robotics. Doctoral Thesis - KTH Royal Institute of Technology. www.diva-portal.org/smash/get/diva2:1078940/FULLTEXT01.pdf. [Online; accessed Dec 08th, 2018]. 2017.
[3] Delaunay Triangulations. https://www.ti.inf.ethz.ch/ew/Lehre/CG13/lecture/Chapter%206.pdf. [Online, accessed Mar 07th, 2019]. 2018.
[4] Continuous Shading of Curved Surfaces. https://ieeexplore.ieee.org/document/1671906. [Online; accessed Aug 09th, 2019]. 1971