# Integration of RC Vehicles in a Robotic Arena

Nuno G. P. Martins
nuno.martins@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2016

**Abstract**

This work is integrated in a group effort to produce an architecture capable of being used by students, for a practical implementation of attitude and position control models, whether it be for a single or a multiple agent environment. Integrated solutions that allow for the use of commercially available Radio Control (RC) vehicles under a new concept of Robotic Arena, in GPS denied environments, were developed. This architecture was designed with focus on replicability, versatility and reliability while aiming for a low-cost solution. Two different approaches were taken, one based in an Arduino, and another based in Raspberry Pi. These different approaches were developed to accomplish the same purposes, but with the potential to provide alternative capabilities. Once the pre-requirements of the system were established, additional hardware was integrated so as to respond to the needs and develop individual solutions. Some of the capabilities that must be ensured are: the communication with the vehicle, the interaction with the vehicle's actuators and recalling information about the vehicle's attitude using suitable sensors. In the end, all these individual components were integrated in a command system, creating two different approaches to handle with the vehicle and include them in the global system. A Simulink model capable of integrating the control blocks, to be produced by the students, was created to allow for interaction with the vehicle and its systems. This work results in two fully usable independent architectures, ready to be replicated and used by students in practical Control Classes.
**Keywords:** Integrated Architecture, Robotic Arena, GPS-denied Environments, Remote Communication, RC Vehicles.

## 1. Introduction

Nowadays with the scientific progress in the most different fields like control, artificial intelligence and robotics, the importance of multi-agent environments becomes a reality that's increasingly compelling. These environments have proved to be very versatile. Applications are varied with cooperation between agents being used to perform tasks that cannot be performed by a single agent or to speed up execution time and improve performance. Independently of the agent's goal, an important part of its organization within a populated environment is the self-awareness of their location and the awareness of the other agents' locations. To accomplish this in GPS denied environments, a variety of sophisticated systems are available on the market but require significant investments. These solutions are generally associated to the use of Robotic Arenas and advanced optical motion capture systems, which are already in use in state-of-art robotics research centres. An alternative approach to produce a new generation of Robotic Arenas is required, to extend the utility not only to the researchers, but to the students as well.

## 2. Overall Vision

The purpose of this work is to develop an architecture that allows for the integration of commercially available RF vehicles in a Robotic Arena. The overall concept of this architecture is based on the idea of using a computer running a Simulink environment to remotely control and command one, or multiple ground vehicles. This computer must receive as input the position of the vehicle provided by a localization system (the proprietary localization system ADIS is in development to be used for this purpose). The computer must be able to wirelessly transmit the actuation commands to the vehicle's actuators and receive the information provided by the set of suitable sensors to be installed on-board the vehicle. This concept asks for the installation of some kind of hardware in the vehicle, in order to promote the interaction with the vehicle's actuators, to handle with the on-board installed sensors and to perform the communications itself. The developed architecture will be referred to as Command System.

Besides all the features already mentioned, the Command System was designed based on some more pre-requirements:

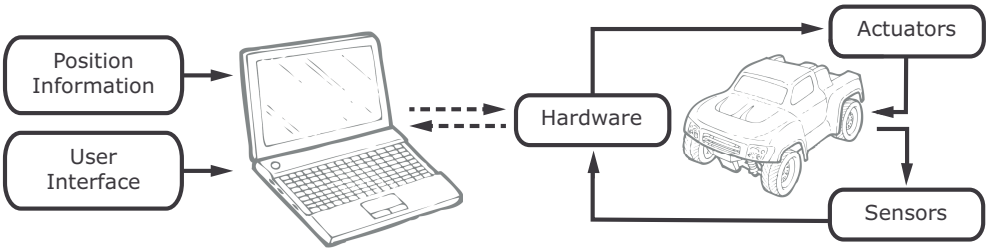


Figure 1: Overall Concept of the developed Architecture.

1. The vehicles must be integrated in the architecture without having their original structure changed;
2. The architecture should be modular and easily replicable;
3. The architecture should allow flexibility in a number of vehicles to be commanded;
4. Working frequency above 30Hz. This corresponds to the working frequency of ADIS. The command system should not degrade the overall system's performance;
5. Keep costs reasonably low throughout the whole platform.

## 3. The RC Model

The model chosen to integrate the architected system was the LaTrax SST, built by Traxxas. It is a 1/18 scale model of a 4WD SST race truck [1]. The SST has an electrical motor powered rechargeable battery. The electrical motor integrates a brake system that is a very useful feature. Capable of reaching a top speed of 12 meters per second and to perform aggressive accelerations, this car is a very good model to accomplish some of the more complex move sequence, like drifts, spins and j-turns. The high-quality construction materials, and the robustness of the structure makes this model very resistant to potential impacts and crashes. The modularity of the LaTrax SST ensures the possibility of replacing any damaged part, without a loss of the whole vehicle. Despite all these features, the cost is very reasonable when compared with similar models. For all of this, the LaTrax SST is a great model to use in a laboratory and to perform the required mission.

### 3.1. Vehicle Setup

To be possible to remotely control the car, it is necessary to know how its actuators work. The first thing that is important to understand is how the original setup is conceived. For this, a list of the original vehicle's components is presented, and a schematic helps to understand the vehicle's setup (see figure 2).

List of components:

- Brushless throttle motor;
- Brushless motor's Electronic Speed Controller (ESC);
- Servo direction motor;
- 6 cell 2/3A NiMH battery;
- RF receiver.

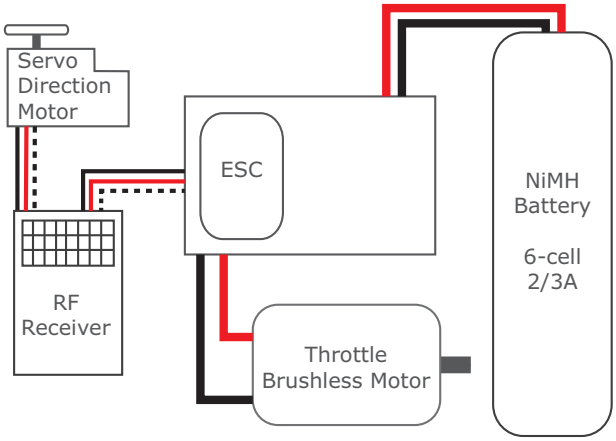


Figure 2: Original vehicle setup.

The connections with the RF receiver are ensured by cables with three wires. These three wires have a colour code that help to understand their function.

- The black wire is ground.
- The red wire is the voltage (ESC: 5V output; Servo: 5V input).
- The white (dashed black in figure 2) wire is the control signal.

Once the RF receiver's outputs are directly connected to the actuator's white wires, if these output signals are known, it is possible to emulate them to actuate the motors.

### 3.2. Vehicle Actuation

According to the manufacturer, the control pin's output from RF receivers provide a Pulse-Width Modulated (PWM) signal, which contains the information transmitted from the transceiver by Radio Frequency. Each channel outputs a PWM wave to actuate one different motor. An oscilloscope allows the visualization of the output signal of one of the receiver's

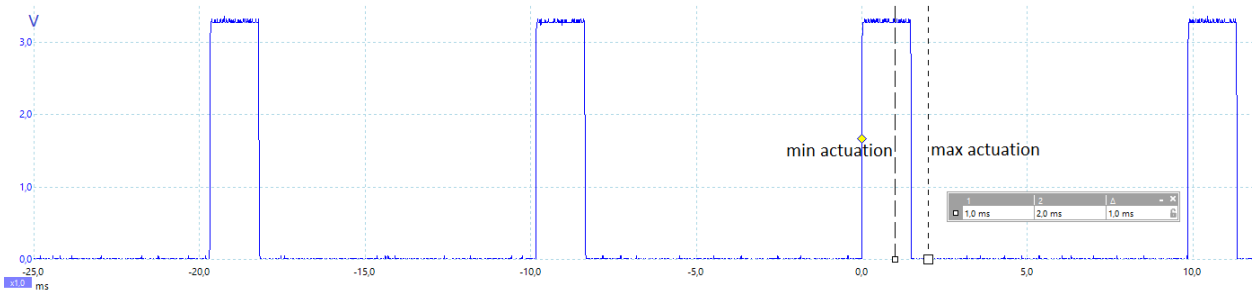


Figure 3: Original PWM signal on the RF receiver's output, no actuation required

Table 1: PWM signals' parameters.

| | Minimum value | Mean value | Meanvalue |
|---|---|---|---|
| **Actuation Required** | 100% reverse | 0% | 100% |
| **High Pulse Width** | 93 $\mu s$ | 1487 $\mu s$ | 1979 $\mu s$ |
| **Duty Cycle** | 10% | 15% | 20% |
| **Period** | | 9840 $\mu s$ | |
| **Frequency** | | 101.6Hz | |

channels, when there is no actuation required on radio transceiver (figure 3).
The vertical dashed bars on figure 3 mark the limits of a maximum and minimum High Pulse Width, when 100% actuation is required and 100% reverse actuation is required in these specific channels. Using the oscilloscope tools, it is possible to characterize the output PWM wave. The PicoScope software allows for the determination of important parameters that help in this characterization. These parameters were determined for each one of the three mentioned states and registered in table 1.
After studying the output of the two receiver channels, it is clear that the PWM signal presents the same behaviour in both. The actuation of the vehicle can be done by generating two different PWM signals, one for each actuator. Once the receiver's outputs are totally known, it is possible to emulate them. To guarantee that no interference that causes an undesired behaviour of the vehicle happens, this emulation of signals must be ensured by a specific hardware that produces hardware PWM signals.

### 3.3. Communication with the Vehicle

As an RC car, the LaTrax SST scale model is equipped with a RF system that guarantees the communication between the user and the vehicle. Despite this being a very reliable technology, it is not usable in this architecture, because their functionalities are out of the scope of this work. Therefore, an alternative is needed to establish a new communication's system. Since the ADIS infrastructure is being designed to use a Wi-Fi network to broadcast the information about the vehicle position, it seems a logical move to use and potentiate this available resource. To perform vehicle control through a wireless communication, it should be guaranteed that the frequency of communication is good enough. The communication itself represents the bottleneck of the system's work frequency. Therefore, it is extremely important to choose the proper protocol for the transport layer of the IP network. It must be assured that the communication's frequency is above the pre-required value of 30Hz.
Since the communication speed is the most valued feature, the implemented communication protocol will be the User Datagram Protocol (UDP), instead of the typically used Transmission Control Protocol (TCP). TCP's protocol reliability is higher due to its ability to prevent the loss of transmission data and ensuring the correct order of packet reception, but it is also quite slower than the UDP protocol. However, it is important to understand that the speed performance of UDP is achieved at the expense of not having the same reliability as TCP.

### 3.4. Vehicle Instrumentation

An important part of the developed architecture is the instrumentation of the vehicle. The ADIS system will provide information about the spatial location of the vehicles, but this information is not enough to know about its spatial orientation. An Inertial Measurement Unit (IMU) is the proper instrument to collect information about the vehicle's attitude. The IMU used was the 10 degree of freedom (DOF) GY-80 model, which integrates the set of sensors shown in table 2. Note that an additional IMU DOF is provided by the BMP080 sensor pressure not used in this work.
The data of all these sensors can be combined to provide a more reliable information about the vehicle's attitude. The GY-80 supports Inter-integrated cir-

Table 2: GY-80 IMU's sensors. LsB means Less Significant Bit.

| Sensor | Model | DOF | Digital Output Resolution | Physical Quantity | Units | Used Range | Scale Factor |
|---|---|---|---|---|---|---|---|
| Accelerometer | **ADXL345 [2]** | 3 | 11 bits | Specific Force | g-force [g] | ±4g | 3.9 mg/Lsb |
| Gyroscope | **L3G4200D [3]** | 3 | 16 bits | Angular Rate | degree per second [dps] | ±2000 dps | 70 dps/Lsb |
| Magnetometer | **HMC5883L [4]** | 3 | 12 bits | Earth's Magnetic Field | Gauss [G] | ±1.3G | 0.92 mG/Lsb |

cuit ($I^2C$), a multi-master serial bus, to configure all the sensors and collect their raw data measurements. To choose the appropriate range for each sensor, a simple test was designed: a preliminary test must be performed using the measurements ranges defined by default. The car must be driven manually using a RF controller, pushing the actuators to their limits, and forcing the car to perform the most aggressive manoeuvres. In this way, it is possible to guarantee that the sensors will be submitted to the worst possible scenario. The collected sensor data can be plotted over time, to see if the sensors' measurements saturate and if so, the sensors ranges must be adjusted. Then, the tests must be repeated, until a satisfactory result is found. As result, it is possible to get the ideal measurement range to use for each sensor. These values are shown in table 2.

## 4. Architecture Development

To approach the problem of the development of an integrated architecture that accomplishes all the established pre-requirements, appropriate hardware must be used. The hardware's features must include hardware PWM generation and two different communication protocols: UDP to communicate with a remote PC and ($I^2C$) to communicate with the IMU. It was decided to produce two different implementations, based on two different approaches. Each one of the approaches, based in different types of hardware, was designed to accomplish the same objectives, but both present different prospects of future development. The differences between the two approaches will be discussed in the upcoming sections.

### 4.1. Solution based on Arduino UNO

To generate the hardware PWM waves in Arduino, one of the timers of the Arduino's microcontroller must be allocated to this functionality. For this, the open source TimerOne library provides a collection of routines that allow for a simple configuration of Timer1, and offers a way to generate these kind of signals. However, this library is not provided by the Arduino IDE, it is available in [5] and must be manually installed. This library is limited as it is only capable of generating the desired signals in pins 9 and 10. Nevertheless, this is enough for the scope of this work. The Arduino Uno board does not support Wi-Fi communication. As so, a Wi-Fi shield is required. The shield used was the TinySine WIFI shield [6]. This equipment provides a way to communicate to a local network by IP address, but the communication between the shield and the Arduino must be guaranteed through Software Serial communication. For this purpose, the Arduino IDE provides the SoftwareSerial library. The maximum baud rate supported by the library is 115200 bps, according to the official documentation [7]. This shield must be manually configured. As a result of this work, a user manual that explains this has been written. The Arduino IDE wire library allows for the access of all of the IMU sensors. By writing in specific sensors' registers, it is possible to choose the measurement ranges and other configurations. By reading other registers, it is possible to collect information about the sensors' measurements. The value of each DOF, of each sensor is outputted by two registers, one referent to the Less Significant Bit and another to the Highest Significant Bit. This last value must be 8 bits shifted left, and the result must suffer a logical disjunction with the LsB value. The resultant binary value must be converted into an integer, and multiplied by a scale factor to give a value with physical meaning. The solution is integrated, and boxed in a 3D printed case, to confer durability to the implementation, and presented to the users as a single component that is easily plugged into the vehicle. The final architecture provided from this approach is represented in figure 4.

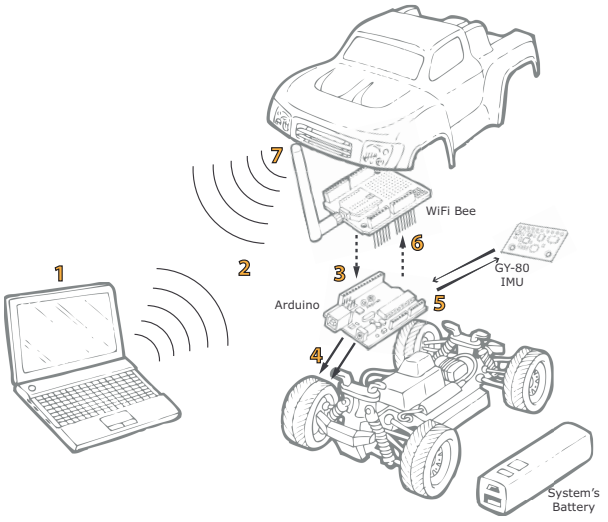


Figure 4: Arduino Based Solution final setup

Figure 4 illustrates the following steps:

1. A computer, running a Simulink solution, must be integrated in the same IP network as the Wi-Fi shield.

2. The computer is able to send UDP command to a target IP. The Dynamic Host Configuration Protocol (DHCP) of the Wi-Fi shield must be deactivated, and their IP must be set.
3. The UDP command message contains information about the PWM actuation signals. This message is received by the Wi-Fi Shield and passed to the Arduino trough Software Serial Communication.
4. In the Arduino loop, this message will be received and if it is correct, the PWM commands' information will be given as input to the actuators, through the wire connections.
5. After the actuation, the Arduino sends $I^2C$ commands to each IMU sensor asking for measurements, and listens to the sensors' measurements from the $I^2C$ port.
6. The sensors' data is converted into a decimal base, and rearranged as a string. This string is passed to the Wi-Fi shield using the SoftwareSerial port.
7. The shield sends the sensors' raw measurements to a target IP. The target of this information must correctly be defined by the user in the shield's configuration.
8. The sequence of actions will be repeated in the loop as the vehicle is shut down.

### 4.2. Solution based on Raspberry Pi

The Raspberry Pi is a small sized computer and, as such, it runs a full-blown operational system. Therefore, as opposed to what happens in the Arduino, it is possible to choose what programming language to use to implement the individual solutions. For the purposes of this work, the Python language was used, due to its extensive libraries and tool boxes. The pigpio library provides a way to generate hardware PWM waves. This is an open source library written in C that relies on a Python module that allows for the control of the General Purpose Input Outputs (GPIO) pins of the Raspberry. This resource and its respective documentation are available in [8]. The Raspberry Pi3 hardware already has Wi-Fi technology incorporated, so no supplementary hardware pieces are needed to implement UDP communication. To be able to implement this protocol, the socket library must be used. This is a regular Python library, installed by default. To send and receive messages, two different sockets must be created, and the reception socket must be attached to the Raspberry Pi's IP and the local receiving port. To deal with the GY-80 IMU implementation, there is an open source library available [9]. This library can retrieve the IMU's measurements, convert them into physical units and process them to get additional information. For this work's purposes, some modifications to the original library are needed. As there is a need to keep the system universal to potentiate the most diversified uses, it was defined that the output of the instrumentation subsystem must be the sensors' raw data. It will allow the user to process data in the most convenient way to attend to their own purposes. As in the Arduino solution, the solution is integrated, and boxed in a 3D printed case, to confer durability to the implementation, and is presented to users as a single component that is easily plugged into the vehicle. The final architecture obtained through this approach is represented in figure 5.

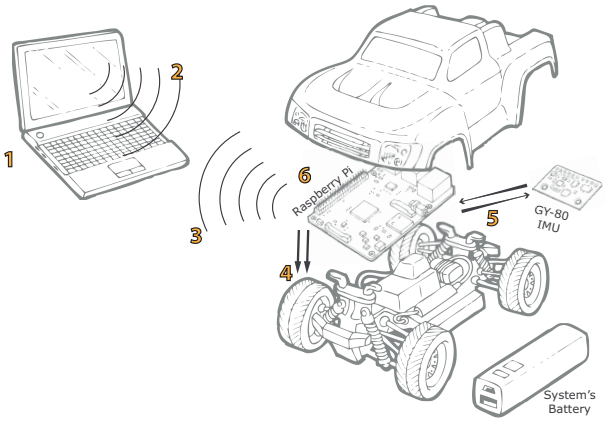


Figure 5: Raspberry Pi Based Solution final setup

Figure 5 illustrates the following steps:

1. A computer running a Simulink solution must be connected to the same IP network as the Raspberry Pi.
2. The computer is able to send UDP command to a target IP. The user must guarantee that the Raspberry Pi's IP is defined to be static.
3. The UDP command message contains information about the PWM actuation signals. The messages are received in the input socket, from where they must be read. These command values must be scaled after being received.
4. The Raspberry Pi actuates each one of the commands just by redefining the duty cycle for each one of the pigpio classes.
5. An $I^2C$ interface is used to communicate with each one of the IMU's sensors and collect their measurements. This functionality is implemented by the IMU_setup library.
6. The sensors' data is converted into a decimal base, and rearranged as a string. This string is written in the output socket, to be communicated to a target IP.

### 4.3. Simulink and User Interaction

The requirements of the system predict the use of a Simulink environment to provide to the user a possibility to interact with the system. As so, a Simulink model was designed, to facilitate the system's usability. The Simulink implementation will be divided in 3 main blocks: The user interface, the setup block and the communications block 6. The user interface allows for an easy use of the whole system. It outputs the commands in a +/-1 range, where 0 in no actuation required, 1 is maximum actuation required and -1 is full reverse actuation required. This interaction can be done just by inputting the values with a keyboard, by using a joystick or by using an interactive dash board panel. It is in this block that the users can integrate their own controller block, as long as they always respect the output scale. The setup block is the one responsible for converting the -1 to 1 range from the user's interface to a range capable of being communicated to the communication block. The setup block also provides a graphical way for the user to choose the desired solution (Arduino or Raspberry), and how to interact with the system (giving all the available options in the user interface). Based on the user's choices, the setup block decides what scale change to apply to the user interface's output, and defines the correct IP target, transmitting it to the communication block. As for the communication, the UDP send block is provided by Simulink and is available within the DSP system toolbox. This block is easy to configure and use. The UDP block is only capable of sending unsigned 8 bit integers. This represents a limitation in the commands allowed for communication, and requires a scale change process in the setup block. Figure 6 illustrates this process.

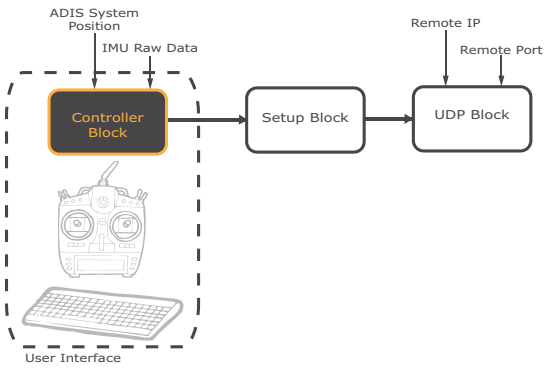


Figure 6: Insertion of the Controller in the Loop, using the Simulink environment.

## 5. Results of the Implementation

### 5.1. Results of the individual solutions

To validate the quality of the hardware PWM signal generated in each solution, an oscilloscope was used to visualize and determine the parameters of the output signals. The procedure is the same as in section 3.2. The results can be seen in table 3. The first consideration about the results is that in both solutions the original wave presents a slightly different behaviour. Both implementations were designed to produce a 10000 $\mu s$ period wave. The results are very similar to the intended ones, and in the both cases the adopted solution provides a fully usable method to actuate the motors.

To validate the UDP communication systems, a similar test was performed for both solutions. This test consists in communicating a sizeable message, always in the same format, but numbered by order of communication, from the integrated solution to a remote PC. In the PC, the messages were received during a certain period and saved in MATLAB. Once the test was finished, the messages received were processed to determine the communication frequency and the percentage of lost messages. The results are summarized in table 4. In the raspberry Pi test, a delay was inserted on the script to limit the maximum allowed frequency to 50 Hz. This decision was taken because, without this imposed limitation, the high communication frequency increases the percentage of lost messages. In the tests that were performed with this limitation fixed at 100Hz, the percentage of lost messages is close to 5%. The Arduino maximum allowed frequency occurs for the 115220 bps baud rate of the Soft Serial communication (results in table 4). As such, the 50Hz limit value was chosen to get similar results in both solutions and to allow for the comparison of both architectures. Analysing the results in the table 4 it is possible to see that the worst-case scenario occurs when the Raspberry Pi reaches a percentage of lost messages above 2.5%. However, this is within the requirements which means that this system is operational to be used.

To validate the IMU measurements, two different tests were performed with the IMU on-board the vehicle. In test 1, the car was at rest during 60 seconds. In test 2, the car was commanded to perform circular movements, always in the same direction, at a constant speed and with a constant radius, over 20 seconds. During this time, the system was used to collect data from the IMU and send it to the remote PC, where the data was processed and plotted over time. In test 1, it is expected that the action of gravity on the car can be easily identified in the accelerometer's measurements. Since the vehicle was stopped, gravity is the only force that affects the sensors and must be sensed in the z-axis, with a value close to 1g. See figure 7. The expected values for the gyroscope must be near 0 dps in all three axes as the vehicle as no angular rate (see figure 8). Finally, the magnetometer's readings must be constant in time, since the relative position of the car to Earth's magnetic field does not change (figure 9).

Table 3: Results of hardware PWM waves generation

| | Original PWM | Arduino PWM | Raspberry Pi PWM |
|---|---|---|---|
| **Frequency** | 101.6 Hz | 99.96 Hz | 101 Hz |
| **Cycle Time** | 9846 $\mu s$ | 10000 $\mu s$ | 9900 $\mu s$ |
| **Duty Cycle** | 15.21% | 15.16% | 15.76% |
| **High Pulse Width** | 1505 $\mu s$ | 1526 $\mu s$ | 1563 $\mu s$ |

Table 4: UDP communication results

| | Raspberry Pi | | Arduino (115200 Baud rate) | |
|---|---|---|---|---|
| | **Frequency** | **Percentage Lost Messages** | **Frequency** | **Percentage Lost Messages** |
| **10 seconds** | 48.8 Hz | 1.02% | 52.9 Hz | 0.38 % |
| **1 minute** | 48.9 Hz | 0.99% | 52.8 Hz | 0.5 % |
| **5 minute** | 48.7 Hz | 1.49% | 52.5 Hz | 0.65 % |
| **10 minute** | 48.2 Hz | 2.61% | 52.5 Hz | 0.94 % |

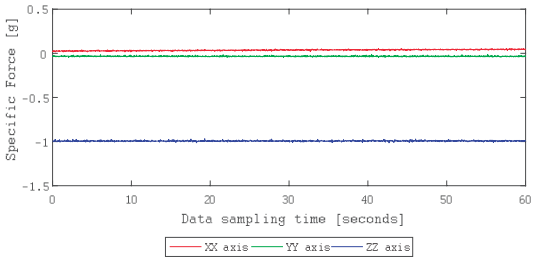


Figure 7: Test 1: Accelerometer

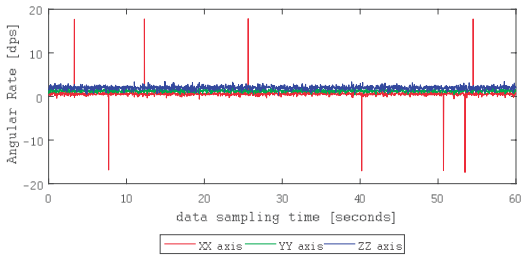


Figure 8: Test 1: Gyroscope

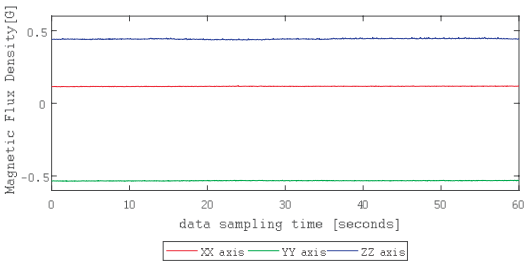


Figure 9: Test 1: Magnetometer

The results of test 2 are displayed in figure 10 and 11, for the gyroscope and magnetometer, respectively.
In the magnetometer's readings, the x and y measurements present a sinusoidal behaviour. Because the vehicle performs circular moves, the sensor's position relatively to the Earth's magnetic field is periodically repeated and, as such, the measured values will be periodically repeated too (one period corresponds to one complete lap).

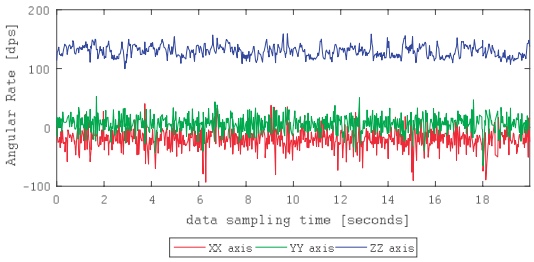


Figure 10: Test 2: Gyroscope

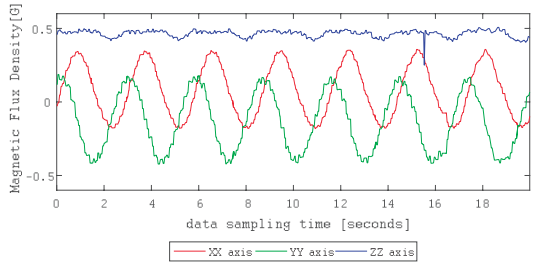


Figure 11: Test 2: Magnetometer

As for the gyroscope, because the vehicle has always the same orientation in the z plane (assuming no ground inclination), it is expected that z measurements should be constant. This is verified as the measurements in the z-axis vary a lot less than the other axes' measurements. The gyroscope's measurements also allow for the identification of the circular movement, but there is no way to guarantee with a complete degree of confidence that the IMU is perfectly horizontal when placed in the vehicle. As the vehicle performed circular movements on a constant z plane (the ground), it is expected that the angular

Table 5: Comparison of the work frequencies of both solutions.

| | **Arduino Operation's Frequency** | **Raspberry Pi Operation's Frequency** |
|---|---|---|
| **10 seconds** | 32.20 Hz | 33.40 Hz |
| **30 seconds** | 32.56 Hz | 33.70 Hz |
| **1 minute** | 32.95 Hz | 33.18 Hz |
| **10 minutes** | 32.33 Hz | 33.38 Hz |

rate measured in the z-axis would be larger than that of the remaining axes, and it is. The mean value of the z-axis angular rate was determined as 129.4510 dps, from the data presented in figure 10. Another method to perform a rough calculation of the mean angular rate about the z axis is to determine the number of completed laps by observation of figure 11. Each lap is equal to modifying the angular position by 360 degrees, and it is possible to see that the angular position varied a total $7 \times 360$ degrees in 20 seconds (the vehicle performed 7 laps). So, a good estimate for the angular rate is $\frac{(7\times360)}{20} = 126\,dps$, which is very close to the mean value of the gyroscope measurements. In this way it is possible to validate the use of all the three sensors in the system. However, it is possible to identify some noise mainly in the graphics related to the test 2, where the motor is working. Despite this, the data from these sensors are perfectly usable, but requires filtering. The system's users will need to deal with that, according to their intentions.

### 5.2. Results of the Overall Architecture

With both architectures fully developed, it is important to compare them and analyse their performance. Several tests allow for the determination of the system's work frequency. This is presented in tabl 5.
The frequency values in table 5 already include the IMU raw data processing in the remote PC.
The first comment is about the work frequency. It is possible to see that the requirements are accomplished as the work frequency is always above the 32 Hz in all the performed tests (30 Hz was the prerequired value). The results from the Raspberry Pi based solution are marginally better than the Arduino Based solution. Detailed tests reveal that the processes that receive the UDP commands are the most delayed processes in the Arduino architecture (limited by the Software Serial communication), while in the Raspberry architecture, the most delayed process is the access to the IMU sensors. These processes have margin for improvements, although there is no benefit in improving the work frequency, because at high frequencies the UDP communication losses increase considerably.
With respect to hardware capabilities, some considerations must be performed. Due to the fact that the Raspberry Pi already supports WiFi features, no additional modules are required like in the Arduino approach. It makes the setup of the Raspberry approach easier, and the solution itself lighter and smaller.
The processing capabilities of the Raspberry Pi are better, however, to accomplish the purpose of this work, the Arduino microcontroller performance is considered to be enough. This specific difference between the two approaches is directly related to the option of developing the two alternative solutions. A bigger processing capability allows for a different implementation possibility. By integrating the controller in the Raspberry Pi's internal script, instead of a Simulink running on a remote PC, it is possible make the vehicle independent of the remote PC. It represents an increase of the system's autonomy.
The number of channels where hardware PWM can be generated is significant larger in Raspberry Pi, where all the 13 GPIO pins provide this feature, against the 2 digital pins (9 and 10) on the Arduino solution. With 2 different waves it is possible actuate 2 different motors, and it is enough for the vehicle used, but can represent a limitation, if there are prospects to use the system to integrate more complex vehicles.
The Raspberry Pi, being a computer, requires the use of computer peripherals, specifically a screen must be used for initial configuration and OS installation. It is possible to use a remote desktop software to remotely get access to the Raspberry's OS, through another computer or a smartphone. The Raspberry also requires more input current, which means less autonomy if equal batteries are used in the two solutions.
The costs of developing each solution are very close and can not be considered to be a tiebreaking factor.

## 6. Conclusions

As a result of this work, there are two independent architectures that allow for an integration of commercially available RF vehicles in a new concept of Robotic Arena. Despite the fact that each architecture was aimed at the same purposes and was developed to accomplish the same pre-requirements, both present different growth prospects. An Arduino based approach aims for a more user friendly solution but a Raspberry Pi based approach allows the user to remove the remote PC from the equation and provides the vehicle with greater autonomy.
During the development stages of this work, a pre-

liminary version of the architecture developed was used by a pair of students during practical Optimal Control classes, to design a position controller that commands the vehicle to perform a pre-established route. It was an integral part of their final assessment and was the ultimate proof of concept and a validation of the systems' utility.

## References


[1] Traxxas Way. *LaTrax SST Owner's Manual, MODEL 76044-1*. Traxxas Way, McKinney, Texas 75070, 2015.

[2] Analog Devices. Digital accelerometer adxl345, 2015. original document from Analog Devices.

[3] STMicroelectronics. L3g4200d mems motion sensor: ultra-stable three-axis digital output gyroscope, 2010. original Preliminary document from STMicroelectronics.

[4] Honeywell. 3-axis digital compass ic hmc5883l, 2013. original document from Honeywell.

[5] Arduino official website. `https://www.arduino.cc/`. Retrieved on February 2016.

[6] Tinysine Electronics. *TinySine WIFI Shield User Manual, version 1.0*. Tinysine Electronics, 2013.

[7] SoftwareSerial library's official documentation. `https://www.arduino.cc/en/Reference/SoftwareSerial`. Retrieved on March 2016.

[8] pigpio library. `http://abyz.co.uk/rpi/pigpi`. Retrieved on june 2016.

[9] Open source python library to IMU-GY80. `https://github.com/peterjc/longsight/blob/master/gy80.pyo`. Retrieved on june 2016.