



COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

IoT Racetrack

Autor: Fidel Sanz Azuara

Director: Álvaro Pérez Bello

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

IoT Racetrack

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2019/2020 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.



Fdo.: Fidel Sanz Azuara

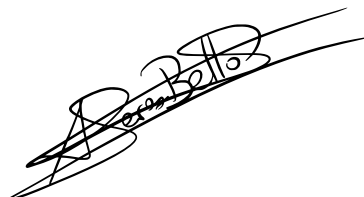
Fecha: 21/ 07/ 2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Álvaro Pérez Bello

Fecha: 21/ 07/ 2020





COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

IoT Racetrack

Autor: Fidel Sanz Azuara

Director: Álvaro Pérez Bello

Madrid

Agradecimientos

Quisiera provechar la oportunidad para agradecer a las personas que me han ayudado tanto durante el grado como en este proyecto.

En primer lugar, a mi madre y a mi padre que me han apoyado en todo momento.

Igualmente, agradecer a mi familia por mostrarme siempre su interés y apoyo.

También agradecer a mis amigos y compañeros del grado con los que tantos buenos momentos he vivido a lo largo de estos años.

Asimismo, me gustaría agradecer a los profesores del grado por la formación que me han proporcionado. En especial a Aurelio, coordinador de este proyecto.

Finalmente, agradecer a mi director de proyecto Álvaro Pérez Bello que tan bien me ha guiado y aconsejado a lo largo de todo el proyecto y me ha facilitado desarrollarlo en colaboración con Altair.

IOT RACETRACK

Autor: Sanz Azuara, Fidel.

Director: Pérez Bello, Álvaro.

Entidad Colaboradora: Altair Engineering Inc

RESUMEN DEL PROYECTO

En este proyecto se ha desarrollado un demostrador de IoT mediante el uso de sensores y dispositivos conectados entre si y a Internet, aplicados a un circuito de carreras a escala. Se muestran así las capacidades del IoT, en especial las de la plataforma de Altair, que ha permitido obtener datos de las diversas fuentes, analizarlos y utilizarlos con diversos fines.

Palabras clave: IoT, MQTT, Gateway, kit de desarrollo, SmartWorks

1. Introducción

Hoy en día se oye mucho hablar, y no solo en el mundo de la ingeniería, sobre el internet de las cosas, en inglés “Internet of Things” (IoT). El IoT se refiere a la conexión de objetos a Internet mediante sensores y actuadores que “permiten conectar el mundo físico con el digital, computadores que permiten procesar esa información y plataformas web donde se procesan y almacenan los datos” [1].

Muchas empresas están desarrollando sistemas/plataformas con las que utilizar el IoT. Para venderlo a los que puedan necesitarlas se necesita mostrar la utilidad del IoT y la facilidad de uso de las herramientas desarrolladas, es ahí donde el uso de un demostrador resulta de gran ayuda, ya que permite observar tanto la utilidad como los requerimientos para aplicarlo.

Este proyecto es una demostración de cómo funciona el IoT a pequeña escala. Para ello se ha montado un sistema consistente en un circuito de carreras a escala que incorpora sensores y un Gateway que recibe, procesa y envía datos interconectando el sistema local y la nube para su tratamiento.

2. Definición del Proyecto

En el proyecto se han planteado diferentes posibilidades u opciones a aplicar en el demostrador. Su posible aplicación se ha evaluado en función de la complejidad técnica y la utilidad para mostrar las cualidades del IoT. Por último, se han desarrollado las que cumplían un cierto nivel de exigencia.

Después, se han planteado los dispositivos necesarios para llevar a cabo el proyecto y los distintos programas a emplear. Una vez se han obtenido dichos dispositivos y el software requerido, se ha procedido a la programación y montaje de todos los

componentes. Finalmente, se ha comprobado el funcionamiento correcto del sistema completo.

3. Descripción del sistema

El sistema consta de un circuito de carreras con su unidad de control, un Gateway, un microcontrolador y dos kits de desarrollo sobre los coches. Al sistema se accede remotamente y se asignan las condiciones de la carrera. Posteriormente mientras la carrera está en marcha se recolectan y muestran los datos recogidos por el microcontrolador, los kits de desarrollo y la unidad de control. Parte del procesamiento de los datos se lleva a cabo en el Gateway y parte en la nube.

En la Ilustración 1 se muestra un esquema de los dispositivos empleados y sus interconexiones.

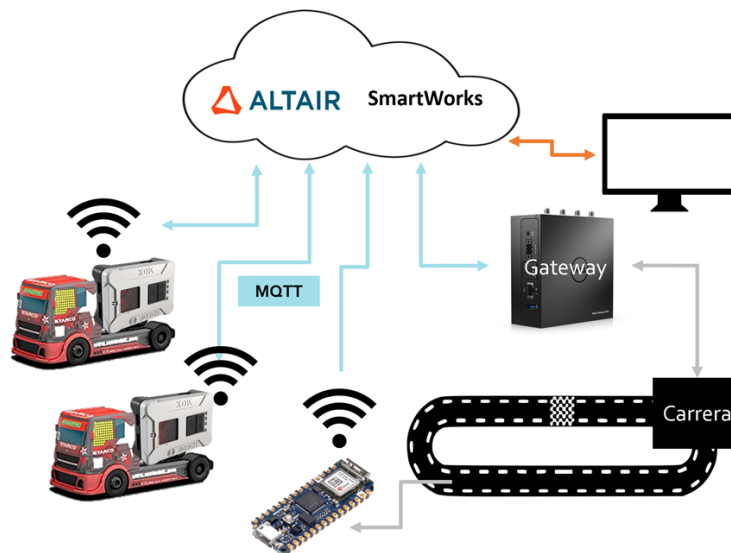


Ilustración 1 - Esquema del sistema

4. Resultados

Tras la programación, interconexión y montaje de los distintos componentes se ha conseguido que el sistema funcione de acuerdo a lo propuesto. Se han podido interconectar todos los dispositivos (sensores, actuadores y puerta de enlace) empleando el protocolo MQTT.

En la Ilustración 2 se muestra parte de la visualización disponible para los usuarios del demostrador. En ella se presentan las aceleraciones obtenidas de los acelerómetros,

las velocidades, el número de vueltas realizadas y la presión aplicada en el acelerador por los conductores.



Ilustración 2: Visualización de los datos

Con ello se comprueba que se ha conseguido obtener los datos de los dispositivos, tratarlos en el Gateway y mostrarlos a los usuarios.

5. Conclusiones

El IoT permite recopilar datos de múltiples fuentes para trabajar con ellos y obtener información relevante para el usuario. En este proyecto se muestra su utilidad en pequeño formato, pero puede ser escalable a muchos niveles lo cual le da mucha versatilidad y permite su aplicación a múltiples disciplinas.

En principio un sistema IoT se puede realizar empleando tecnologías y conexiones disponibles a bajo coste, pero con gran dificultad en su desarrollo pues requiere una serie de conocimientos técnicos y la dedicación una gran cantidad de tiempo, tanto mayor cuanto más extenso se quiera que sea el sistema. Para ello existen plataformas, como por ejemplo SmartWorks, que facilitan la implantación de estos sistemas a cualquier escala, permitiendo desde sencillos desarrollos a complejos sistemas.

Con este proyecto se ha demostrado la aplicación práctica del IoT y se ha dejado abierta la posibilidad de seguir explorándolo en futuros proyectos.

6. Referencias

- [1] Ingeniería Asistida Por Computador, "¿Qué es IoT?", [Online]. Disponible en: <https://www.iac.com.co/que-es-iot/>, [Acceso 05 mayo 2020]

IOT RACETRACK

Author: Sanz Azuara, Fidel.

Supervisor: Pérez Bello, Álvaro.

Collaborating Entity: Altair Engineering Inc.

ABSTRACT

In this project an IoT demonstrator has been developed using sensors and devices interconnected and connected to the Internet, all applied to a slot car racetrack. Showing the capabilities of IoT, specifically the ones from Altair's platform, that enables to obtain data from multiple sources, analyze them and use them for different goals.

Keywords: IoT, MQTT, Gateway, development kit, SmartWorks

1. Introduction

Nowadays the term IoT is used frequently and not only in the engineering world. IoT, which stands for Internet of things refers to the connection of objects to the Internet through sensors and actuators “that allow the physical and digital world to be interconnected, computers/servers that process this information and web platforms where the data is processed and stored” [1].

Many companies are developing systems/platforms with which to use IoT. To sell these platforms to those interested they need to show the IoT usefulness and the ease of use of the developed tools, that is where the use of a demonstrator is very helpful, since it allows people to see both the utility and the requirements to apply it.

This Project is a demonstration of how IoT works in a small scale. For this, a system consisting of a slot car racetrack has been assembled, incorporating sensors and a Gateway that receives, processes and sends data interconnecting the local system and the cloud for its treatment.

2. Project definition

Different possibilities or options to apply in the demonstrator have been proposed in the project. Its possible application has been evaluated based on technical complexity and usefulness to show the qualities of the IoT. Finally, those that met a certain level of demand have been developed.

Then, the necessary devices to carry out the project and the different programs to be used have been proposed. Once these devices and the required software have been obtained, all the components have been programmed and assembled. Finally, the correct operation of the complete system has been verified.

3. Description of the system

The system consists of a slot car racetrack with its control unit, a Gateway, a microcontroller and two development kits on the cars. The system is accessed remotely and race conditions are assigned. Later, while the race is in progress, the data collected by the microcontroller, the development kits and the control unit are collected and displayed. Part of the data processing takes place at the Gateway and part in the cloud.

In Figure 1 a diagram of the devices used and the connections made is shown.

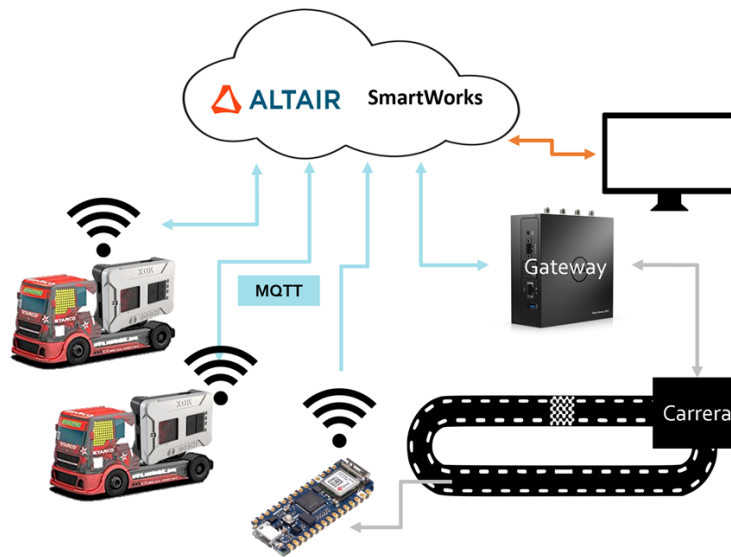


Figure 1: System diagram

4. Results

After programming, interconnecting and assembling the various components, the system has been operated as proposed. All devices (sensors, actuators and gateway) have been interconnected using the MQTT protocol.

Figure 2 shows part of the visualization available to the demonstrator users. It presents the accelerations obtained from the accelerometers, the speeds, the number of laps made and the pressure applied to the accelerator by the drivers.



Figure 2: Data visualization

With this we certify that we have obtained the data, processed it and displayed it to the users.

5. Conclusions

The IoT allows to collect data from multiple sources to work with them and obtain relevant information for the user. This project shows its usefulness in a small format, but it can be scalable to many levels, which gives it great versatility and allows its application to multiple disciplines.

In principle, an IoT system can be carried out using technologies and connections available at low cost, but with great difficulty in its development, since it requires a series of technical knowledge and the dedication of a large amount of time. There are platforms, such as SmartWorks, that facilitate the implementation of these systems at any scale, allowing from simple developments to complex systems to be developed.

With this project the practical application of the IoT has been demonstrated and the possibility of further exploring it has been left open for future projects.

6. References

- [1] Ingeniería Asistida Por Computador, “¿Qué es IoT?”, [Online]. Available: <https://www.iac.com.co/que-es-iot/>, [Accessed 05 mayo 2020]

Index

Chapter 1. Introduction	1
1.1. About Altair	1
1.2. About Smartworks	2
Chapter 2. State of the art	4
2.1. Researched solutions in the industry	4
2.1.1. IoT demonstrators.....	4
2.1.2. Racetrack IoT demonstrators.....	5
Chapter 3. Project definition	7
3.1. Consider functionalities for the demonstrator and study their feasibility.....	7
3.1.1. Monitor car metrics in real time	7
3.1.2. Driver registration	8
3.1.3. Real time information visualization	8
3.1.4. Historic information visualization	9
3.1.5. Monitor real time position	11
3.1.6. Crash detection	12
3.1.7. Race parameter optimization	14
3.1.8. Ghost car	14
3.1.9. Autonomous car	15
3.2. Define establish the final project to be developed	16
3.2.1. Monitor car metrics in real time	16
3.2.2. Driver registration	16
3.2.3. Real time information visualization	16
3.2.4. Historic information visualization	17
3.2.5. Monitor real time position	17
3.2.6. Crash detection	17
3.2.7. Race parameter optimization	17
3.2.8. Ghost car	17
3.2.9. Autonomous car	18
3.2.10. Election of functionalities implemented in the final project.....	18
Chapter 4. System developed.....	20
4.1. Elements of the system	20

4.1.1. Hardware.....	20
4.1.2. Software	22
4.1.3. Protocols and other useful concepts	22
4.2. System instrumentation and obtaining the signal.....	25
4.2.1. Accelerometer	25
4.2.2. Gyroscope	25
4.2.3. Throttles	25
4.2.4. Digital track control unit	27
4.3. Signal processing.....	29
4.3.1. Accelerometer	29
4.3.2. Gyroscope	30
4.3.3. Throttles decoding	30
4.3.4. Digital track control unit	32
4.4. System communication	33
4.4.1. Development Kit.....	33
4.4.2. Arduino.....	34
4.4.3. Digital track Control Unit.....	34
4.5. Design the backend Application	35
4.6. Budget.....	38
Chapter 5. Results analysis	39
Chapter 6. Conclusions and future projects	40
Bibliography.....	41
ANNEXES.....	43
ANNEX I. SDG	43
ANNEX II. Arduino throttles code	45
ANNEX III. Carreralib modified code.....	50
ANNEX IV. Throttle processing	51
ANNEX V. Arduino Nano 33 IoT datasheet	57
ANNEX VI. Bosch XDK 110 Datasheet	59

Figures Index

Figure 1: Altair Engineering Inc. logo.....	1
Figure 2: Altair SmartWorks logo	2
Figure 3: The Smart MiniFAB.....	4
Figure 4: European IoT project	5
Figure 5: Data shown in a monitor in the Qlik demonstrator.....	6
Figure 6: Oracle racetrack demonstrator.....	6
Figure 7: Possible information shown in real time.....	9
Figure 8: Possible historic information presentation.....	10
Figure 9: Race historic information.....	10
Figure 10: Possible general information.....	11
Figure 11: Position representation on a map and on a line.....	12
Figure 12: Vehicle crash.....	13
Figure 13: Ghost car representation.....	14
Figure 14: Control Unit connections and control elements.....	20
Figure 15: Racetrack control unit and trucks.....	21
Figure 16: Microcontroller (Arduino Nano 33 IoT).....	21
Figure 17: XDK-110.....	22
Figure 18: MQTT example.....	23
Figure 19: Racetrack signal [15]	26
Figure 20: Arduino Circuit Sketch	27
Figure 21: Node-RED logo	29
Figure 22: Basic Node-RED flow applied to all signals	29
Figure 23: Throttle flow	32
Figure 24: System communications architecture.....	33
Figure 25: Development kit communication.....	33
Figure 26: Arduino communication	34
Figure 27: Track Control communication	34
Figure 28: SmartWorks visualization of devices connected	35
Figure 29: Data stored in SmartWorks	35
Figure 30: Detail of throttle data stored in SmartWorks	36
Figure 31: Detail of XDK stored data in SmartWorks	36

Figure 32: Real time visualization of measurements	39
Figure 33: SDG 9.....	43
Figure 34: SDG 11.....	44

Tables Index

Table 1: Scores given to technical implementation and to impact for each functionality	18
Table 2: Functionalities Total Score.....	19
Table 3: Throttle parameters.....	31
Table 4: Meaning of throttle parameters	31
Table 5: Hardware budget	38

Equations Index

Equation 1: How to calculate total score	18
Equation 2: Voltage divider.....	26
Equation 3: Speed calculation from acceleration	29
Equation 4: Position calculation.....	30
Equation 5: Angle calculation	30

Chapter 1. Introduction

This project has been developed with Altair Engineering Inc. as collaborating entity for the development of an IoT demonstrator for the SmartWorks platform.

1.1. About Altair

Altair is a “leading provider of enterprise-class engineering software” [1] which enables a faster and cost-efficient development of products through all their lifecycle, from design to operation.



Figure 1: Altair Engineering Inc. logo

Their main software is simulation based. Their suite of software optimizes the product development through multiple programs specialized in multiple disciplines such as structures, motion, system modeling, fluids and thermal management, providing the user with data analytics and truthful rendering.

Altair was founded in 1985 in Michigan focusing on engineering consulting. Later they specialized in simulation software. It is headquartered in Troy, Michigan, and has expanded with more than 40 offices in sixteen countries around the world.

Through the years Altair has acquired multiple software companies to strengthen their simulation capacities offering. It has also partnered with many companies to improve the scope of their business.

Lately, in the last few years, Altair has entered in the IoT services business and it has developed SmartWorks, a platform that is used in this project, described in section 1.2.

1.2. About Smartworks

Altair SmartWorks is a cloud-native platform, which offers an integrated set of services and features to help users to easily connect their things to the digital world, collect their data, manage both, data and devices, and build an app to run it.



Figure 2: Altair SmartWorks logo

Smartworks is available to be deployed on-premises or as a Platform as a Service (Paas), in a private or open cloud. “PaaS is a set of services aimed at developers that helps them develop and test apps without having to worry about the underlying infrastructure” [2].

Altair SmartWorks helps the users execute their IoT projects in an “easy-to-use, reliable, and highly scalable environment” [3].

Altair SmartWorks is part of the Altair SmartWorks™ suite, which is an open-architecture solution enabling advanced edge-to-cloud IoT applications and augmented data analytics powered by machine learning to drive the innovation.

SmartWorks uses thing descriptions defined by the W3C (World Wide Web Consortium) that enables to work with other services and open source projects without the hassle of having to deal with compatibilities.

About the Racetrack demonstrator

As Altair is trying to increase its IoT client base they are developing several demonstrators to show the usefulness of their software and services. This allows future clients to see how they can implement SmartWorks platform to their business.

In this project a demonstrator based on a racetrack is used. It is a good example of the capabilities of the IoT platform because it deals with the instrumentation, communications and data processing in the system.

In this demonstrator the data that can be treated is the car parameters, such as throttle, acceleration, speed, lap count and lap time. Other information can be added like the name of the drivers. This data is sent using wired and wireless connections to a server. Then it is processed in the gateway and in the cloud. The information attained is then stored in the cloud and displayed to the user in a user-friendly interface.

Chapter 2. State of the art

2.1. Researched solutions in the industry

For the research of solutions for IoT demonstrators we will be looking into two kinds of demonstrators. First, general demonstrators that are being used to show the possible usefulness of IoT. Then IoT demonstrators that use a racetrack.

2.1.1. IoT demonstrators

Demonstrators are used to show what a company is offering. They are also a great tool to receive feedback for future product design. Many companies and institutions have made IoT demos to show the possibilities of usage of IoT in multiple environments.

A good example is “The Smart MiniFab”, a project developed in Albert-Ludwigs-University Freiburg. It was developed in order to allow students to work remotely with an already built demo factory. The access was done through a web application. Through this application the user could upload code and receive feedback. What this project demonstrated was how industrial IoT works and which hardware and software it is needed to develop this system [4].

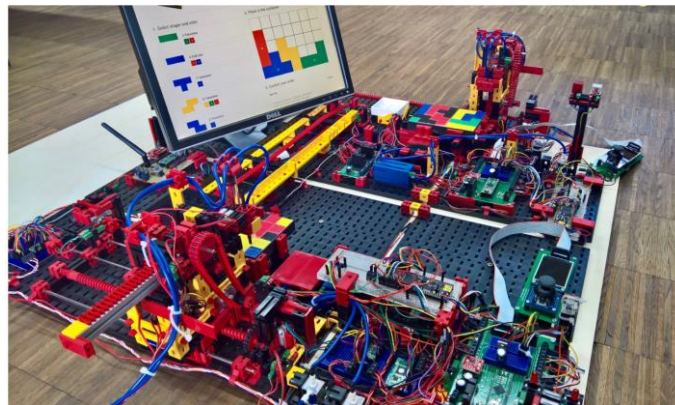


Figure 3: The Smart MiniFab.

PETRAS, which is an institution from the UK that “exists to ensure [...] technological advances in the Internet of Things” [5], and BRE group, which is “a world leading, multi-disciplinary, building science centre with a mission to improve buildings and infrastructure, through research and knowledge generation” [6], developed a “IoT in the Home Demonstrator”. This demonstrator was aimed to explore benefits and create recommendations for secure IoT systems, among other objectives. One of these houses was built and some people were invited to try it and give feedback. The house was equipped with “smart” devices available in the market and some prototypes. Through

this experiment they found some concerns with this kind of services, such as security and privacy issues, that would have to be addressed for future designs [7].

The M2M Industry Working Group was working on an IoT demonstrator using a tabletop greenhouse. The purpose of their connected greenhouse was to report its status (temperature, humidity, luminosity) and being able to control it (turning artificial lighting on/off). They used an Arduino and a Raspberry Pi, as a gateway, to process the data. Three different sensors were used for light, temperature and moisture. They had a servo as actuator to open the greenhouse [8].

A program of the European Union called “Internet of things for European small and medium enterprises” (IoT4SMEs) partnered with universities to develop some IoT related projects. One of these projects was the development of IoT demonstrators. The idea behind the demonstrators they have developed is “to give a real evidence of the potentiality of IoT technologies and to show their potential use in different contexts and applications” [9]. The demonstrators included in their website allow the user to understand the different parts of an IoT platform in an interactive way.



Figure 4: European IoT project

A student of the Open University of Catalonia wrote his end-of-master project on the subject called “Internet of Things demonstrator base on the OpenMote platform”. He did not get to build it, but he developed the system to monitor and make changes in a house. It was aimed at optimizing the energy consumption of the house among other things [10].

2.1.2. Racetrack IoT demonstrators

There have been several companies that have developed a racetrack IoT demonstrator as it is not only instructive but entertaining. This way it is more attractive for visitors in a conference and may attract more possible clients.

Qlik developed what they called “the IoT race game”. They used a racetrack with sensors measuring the number of laps, lap times and engine power. All this data was then showed on a display. They also stored the information in order to see different patterns and behaviours. In order to make it more competitive a quiz was introduced to

determine the maximum speed of the car. The answers in the quiz were scored and they determined the power of the car the users would be driving. The better the people did in the quiz the faster their car could go in the race. When this demonstrator was shown in some events it had great success with plenty of people interested in it [11].



Figure 5: Data shown in a monitor in the Qlik demonstrator.

Oracle made their IoT Demo to show their integrated cloud services. They used a racetrack where all the cars were controlled via Bluetooth. They had various points to prove. First, they wanted to generate a considerable amount of information. To do this they had the racetrack as the source of the data. Secondly, they needed to connect all the information received from the sensors to the cloud. For that they used a Raspberry Pi. Thirdly, their services in the cloud worked with the information to create more data, for example if a car was off the track, they would send an emergency signal. Finally, they had the option to integrate some other services such as voice control or augmented reality [12].



Figure 6: Oracle racetrack demonstrator.

Chapter 3. Project definition

3.1. Consider functionalities for the demonstrator and study their feasibility

The demonstrator is built as a racetrack because it is a very visual way to show IoT working. It is interactive for the possible customer to use and understand. The demonstrator needs to have several functionalities that make possible to sell the capabilities of the IoT system.

In the following sections we describe alternative functionalities. Each of them is evaluated to know which technical implementation they require and the impact they have.

These functionalities, were all of them to be implemented, would make it one of the best IoT demonstrators. Not only this but also it would be possible to apply most of the functionalities to car makers so they can improve their testing. In section 3.2 it is discussed which are implemented and why.

3.1.1. Monitor car metrics in real time

3.1.1.1. Description

We want to monitor acceleration, speed, steering angle, throttle of each car on the track and the number of laps in real time. This information is available to be displayed in real time and stored for future analysis.

3.1.1.2. Technical implementation

To get real time acceleration of the car we need an accelerometer on the car. The steering angle is taken from a gyroscope. The speed is calculated from the data obtained from the accelerometer. To monitor the throttle, we need to get the signal transmitted through the rails. To do so a microcontroller is used. With this measurement it is calculated the amount of pressure that is being made on the throttle. The lap count is carried out by the Control Unit in the track and transmitted to the system through a serial connection. To make the calculations the information has to be sent to the gateway or cloud to be computed.

3.1.1.3. Impact

Knowing acceleration, speed and throttle is basic information, crucial for other functionalities. Among these functionalities, that require these information, are

the determination of the position of the car (section 3.1.5) (which requires speed) and the visualization of these parameters in real time (3.1.3).

To the drivers this information is useful to know how long till the race finishes and where in the leaderboard they are.

3.1.2. Driver registration

3.1.2.1. Description

We want to register each driver in the race. The system asks the driver the name and surname, then all the data that has been collected from this driver in the race can be stored adequately to use in future analysis.

3.1.2.2. Technical implementation

We need a registration process that allows to record the names and surnames of the drivers. The driver must input the information before the race begins in a webpage/excel in a computer. Then a car is assigned to the driver and the system links the name of the driver with the car that he is driving in the race. All this information is stored in a spreadsheet. Internally a number is assigned to each driver as an ID to which the data of their races is logged.

3.1.2.3. Impact

Having a name assigned to each driver allows the system to keep track of the scores and parameters of one driver in many races. This will help to make personalized statistics. As we have the best lap times assigned to the drivers, and some drivers may have more than one, an all-time leaderboard can be made without repeating one driver's time.

3.1.3. Real time information visualization

3.1.3.1. Description.

We want to show on a display all the relevant information in real time. This information is the speed and acceleration of each car, the leaderboard of the race, the pressure applied to the throttle, the position of the cars, a warning if a car crashes and some other related information. This information is shown in a user-friendly way.

3.1.3.2. Technical implementation

We need a screen where the information will be shown. It is connected to a computer which receives the information, collected in real time, to be shown

from the cloud or the gateway. The information is represented with tables and different graphics as shown in the Figure 7, which is a possible solution.

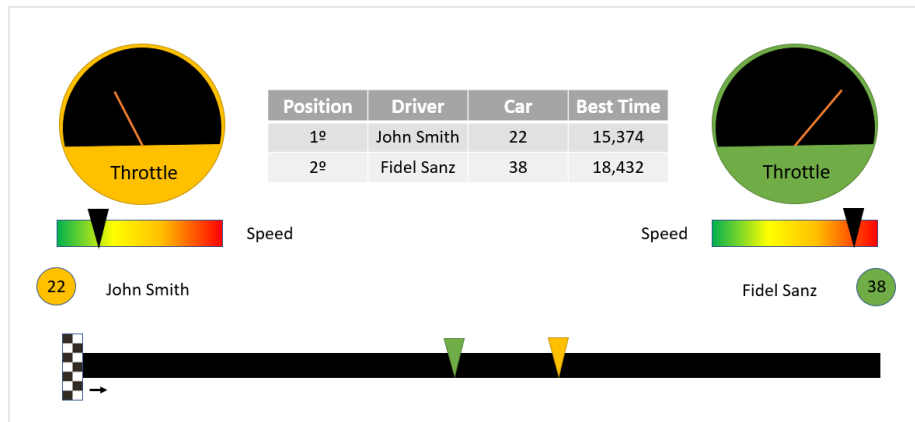


Figure 7: Possible information shown in real time

3.1.3.3. Impact

As this project is a demonstrator, this allows us to show how the real time communication with the cloud and the different devices works. It makes it possible to show the possibilities of the software concerning data visualization.

To the drivers this information helps them improve their lap times and know where in the race and leaderboard they are.

3.1.4. Historic information visualization

3.1.4.1. Description

We want to be able to see information of previous races and the historic of each driver on a display. The information is seen either by driver or by race and a visualization of general information is also possible.

If a driver is selected, we see: number of times he has overtaken another driver and number of times he overtakes another driver, average lap time (of the last race or of all the races in which the driver has participated), average speed of the whole race, number of times the car crashes, a speed heatmap of the best lap the driver has ever made and the positions in which the driver finished all the previous races (Figure 8).

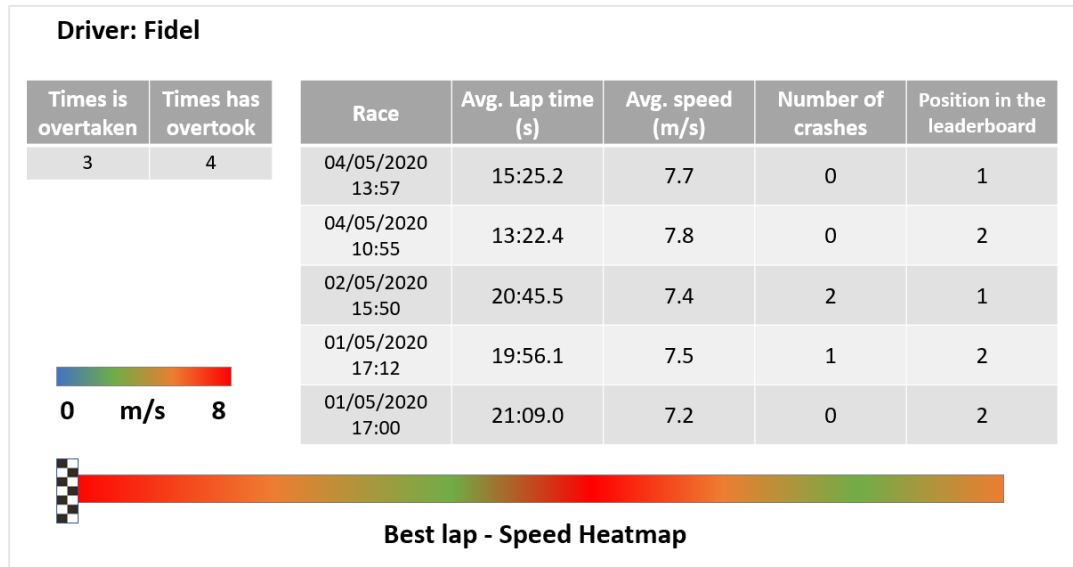


Figure 8: Possible historic information presentation

If a race is selected the information displayed is the number of times each driver has overtaken another, the average lap time of each driver, a speed heatmap of the best driver best lap, the number of times each driver has crashed and an all-time leaderboard (Figure 9).

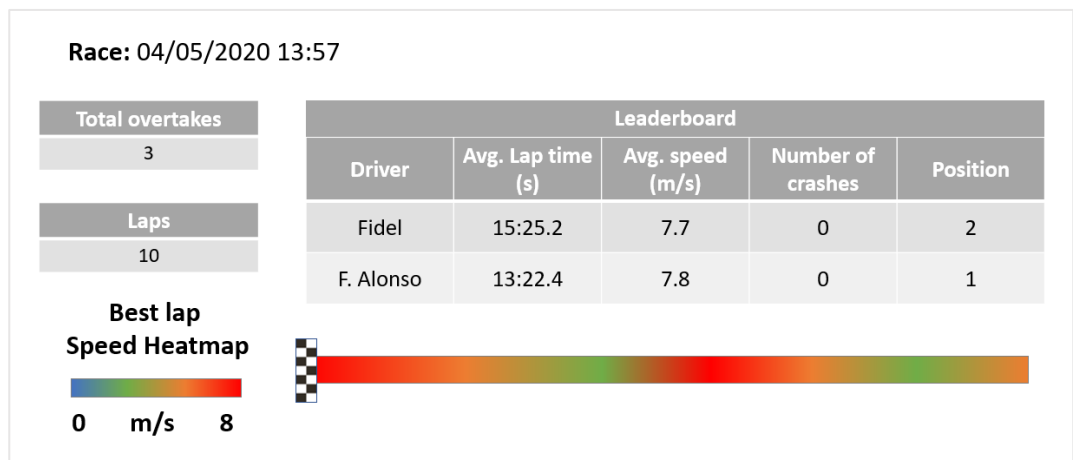


Figure 9: Race historic information

General information view shows the number of races run, the number of drivers in the system, an all-time leaderboard and other statistics (Figure 10).

Total races		Races Today	
21	5		

Drivers		Drivers today	
5	3		

Leaderboard				
Driver	Position in the leaderboard	Avg. speed (m/s)	Number of crashes	Avg. Lap time (s)
F. Alonso	1	8.0	0	13:22.4
Marta	2	7.8	0	14:24.4
Álvaro	3	7.4	2	16:45.5
Fidel Sanz	4	7.1	4	19:56.1
John Smith	5	6.9	0	21:09.0

Figure 10: Possible general information

3.1.4.2. Technical implementation

The information that has been recorded and stored in the races is processed. It is divided into different parameters: times, names, speeds... With these collected parameters analysis and calculus are made. Then a program allows the user to decide which information he wants to access: driver information, race information or global information. Then the program shows a view of the parameters asked for.

3.1.4.3. Impact

This allows us to demonstrate the possibilities of the software concerning data handling and visualization. To automotive industry it could be used when testing cars; with the information they have collected from their tests they can upload it to the program and get the information in easily interpretable tables and graphs.

3.1.5. Monitor real time position

3.1.5.1. Description

Monitor the position of each car in the race. This makes it possible to show on a screen where the cars are at all time (section 3.1.3) and display information when the car is in a certain point in the track.

3.1.5.2. Technical implementation

To get the information of where the car is involves the use of accelerometer and gyroscope data to determine distance traveled and direction in which it has traveled and the parameter of the distance that a car has to go to make a lap. We also need a reference point where to start counting distance; this point is where the laps are counted. The distance travelled is calculated with the speed and acceleration. As calculating speed and distance using only accelerometers and gyroscopes is not reliable enough to get precise values, we need to implement some additional measures. One option to reduce the accumulated error in the distance travelled is to reset the distance each time the car goes through the lap counter. The visualization of this information can be done with a map of the track or with a line representing the distance the car has to travel.

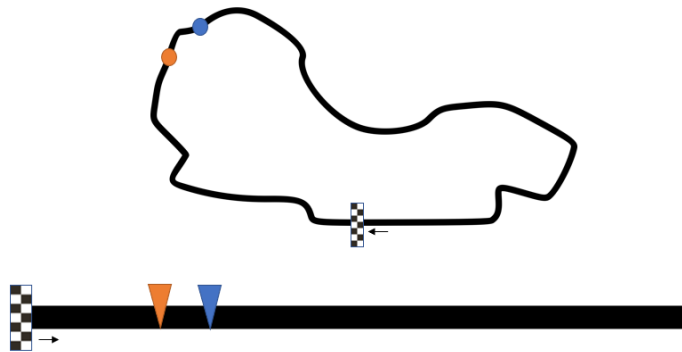


Figure 11: Position representation on a map and on a line

3.1.5.3. Impact

This information is shown on a screen during the race (section 3.1.3) so the drivers and spectators know where the cars are on the track. After the race the information of this functionality plus the information of speeds stored (section 3.1.1) are used to show a heatmap of the speeds during the fastest lap (section 3.1.4) so the driver can see where it is better to drive faster. And combined with the information of the optimum speed (section 3.1.7) we show the driver if he should press the throttle harder or softer to improve his lap time.

3.1.6. Crash detection

3.1.6.1. Description

We want to know if a car has crashed and collect this information. A car is considered to have crashed when it goes out of the track. The place in the track

and the time in the race where the car has crashed are collected each time this event happens.



Figure 12: Vehicle crash

3.1.6.2. Technical implementation

To know if a car has crashed there are several ways to do it. One is to use a gyroscope that detects a sudden movement of the car. Another possibility is to detect if there is pressure in the throttle but no acceleration in the car.

There are various ways to know if a car has crashed. The first one is to use the data received from the gyroscope, if it detects a sudden movement of the car the system says that the car has crashed. This could not be very precise because the car can go out of the track without making a sudden movement, for instance when going fast in the straight line and driving in the tangent to the turn. The second way is to use the position of the car, this is only applicable if the position is known precisely. The third way is to detect if the car does not move when there is pressure in the throttle; this would mean the car is not receiving power from the track therefore the car is out of the track.

3.1.6.3. Impact

With this information we can develop graphs showing how many times a car crashes in a race and where and when it is more common (using sections 3.1.4 and 3.1.5). This is also applicable to the automotive industry. If all cars had a system that detected if they had crashed, they could send a signal for the emergency services to come to assist the injured passengers.

3.1.7. Race parameter optimization

3.1.7.1. Description

This functionality consists in finding the optimum parameters of the car during the race. These parameters to optimize are speed, acceleration and throttle pressure. During the race a symbol indicates if the driver could go faster, should go slower or is driving at optimum speed.

3.1.7.2. Technical implementation

To do this we must calculate the optimum parameters for each point in the track. This could be done mathematically or through a trial and error process with a car on the track checking parameters, maybe with a digital control.

3.1.7.3. Impact

This would allow for other improvements and to show how Altairs programs work using controllers.

3.1.8. Ghost car

3.1.8.1. Description

Display in a map where all the drivers in the race are and add a ghost car. The concept ghost car is used in the race car videogame industry. It refers to a car in a race that does not interfere with the rest of the cars, but it is visible its position on the track. It is mainly used in the races against the clock because the driver does not only want to improve the lap time but also win against the ghost car.



Figure 13: Ghost car representation

3.1.8.2. Technical implementation

To display where the cars are on the track, we need to know their position. We also need some timestamps and positions recorded for the ghost car. With this

information we could show on a display a map of the track and some moving points where the cars would be.

3.1.8.3. Impact

This would be a way of showing how the collected information from previous races could be shown at the same time as the real time information.

3.1.9. Autonomous car

3.1.9.1. Description

This improvement involves a car driving around the track alone at optimum speed. The other drivers can race against this car which is, theoretically, impossible to beat.

3.1.9.2. Technical implementation

This would require different steps. First, we would need to alter the information transmitted through the rails. The information of the speed of each car is transmitted coded through the rails. Then we would have to send the optimum parameters of the throttle, which could already be calculated, to the rails. To do this we need to know where the car is on the track to send the correct parameters.

3.1.9.3. Impact

This would allow the demonstrator to work without human interaction or just with one person driving one car. We could have one car driving autonomously and a person driving another just to show how it works, or we could have some cars driving autonomously at different speeds. These cars would generate data that could be shown on a display or used for analysis.

3.2. Define establish the final project to be developed

This project develops a racetrack IoT demonstrator. It consists on a racetrack with cars, a development kit on each car, a device reading the information from the digital track, some displays, a gateway and a cloud service.

The project incorporates some of the functionalities described in the section Chapter 3. To decide which are implemented a score, from 1 to 9 is given to the technical implementation and the impact. Then the Equation 1 is used to get a total score with which to decide to implement the functionality or not.

Some of the functionalities are not achievable in the time available for this project or exceed the computing capabilities of the devices used. These functionalities could be developed in the future. In the following sections we are going to see the viability of each improvement.

3.2.1. Monitor car metrics in real time

The technical implementation to get this information is relatively easy to apply but it concerns the getting the data and sending it to be computed as fast as possible. Therefore, is not as simple as it seems, and it gets a score of 3.

The direct impact of this functionality is not too big, but it affects the implementation of other functionalities that would be impossible to apply without the data obtained in this functionality. The score given to the impact is 5.

3.2.2. Driver registration

The technical implementation to register the drivers is easy, as it only involves having a menu, and gets a score of 1.

The impact of this functionality gets a score of 1 because although it is useful to make personalized statistics, it is not essential for other improvements.

3.2.3. Real time information visualization

The technical implementation score given to this functionality is 4 because the data has to be transmitted in real time and displayed in an organized way.

The impact score is 4 as it useful for the drivers in the race to know the information available. It also serves to show the capabilities of the system, as this is the most visual part of the project.

3.2.4. Historic information visualization

The technical implementation deals with stored information and with the calculations for analysis. It gets a score of 2.

The impact gets a 2 because it is useful to see this information but is less important than the information provided by the functionality in real time (section 3.1.3).

3.2.5. Monitor real time position

The technical implementation of real time position is complicated to develop as we need information from various sensors as well as some predefined parameters. It gets a score of 5.

The impact of this functionality gets a score of 6 because it can provide useful information to show the user during and after the race.

3.2.6. Crash detection

The technical implementation is complicated because it involves detecting when the car is out of the track as fast as possible without mistaking it for a false accident. It gets an 8.

The impact of this functionality is high as everyone is interested in reducing the amount of accidents both in races and in the road, and when an accident occurs the sooner the help arrives the better for the people in the car. It gets a 9 for the impact score.

3.2.7. Race parameter optimization

The technical implementation involves the use of advanced control theory or a huge amount of data from the cars in the track using a neuronal network. This functionality has enough content to develop to be an individual project, therefore it gets a score of 7.

The impact of this functionality is high because it allows to travel through the track at the highest speed with the lowest risk of accident. The information provided by this functionality permits to use both the ghost and autonomous cars. For all this it gets an 8.

3.2.8. Ghost car

The technical implementation gets a 6 because, although there are other functionalities that provide the data of positioning of racers and the optimum variables of speed for each position, we require to have many functionalities working before being able to develop this. It would get a lower score if the other functionalities were to be already developed

The impact of this functionality is 3 because this functionality is more to enjoy the system as a game than to prove the capabilities of IoT.

3.2.9. Autonomous car

The technical implementation gets a score of 9 due to the difficulty modifying the signal through the track rails. We would need to take the control unit of the track, which is the one that generates the signal, and substitute it by a signal generator which signal is modified following the optimum values of speed. Using this new signal generator, we would have to include in the code the input of remotes if we were to allow drivers to compete with the autonomous car.

The impact of this functionality is 7. As stated in section 3.1.9.3 this makes the demonstrator work without the need of someone operating it, which allows the person selling the product to be able to interact with the clients.

3.2.10. Election of functionalities implemented in the final project

With all this information Table 1 has been created where there is a summary of the grades given to implementation and impact for all functionalities.

Nº	Name	Implementation	Impact
1	Monitor car metrics in real time	3	5
2	Driver registration	1	1
3	Real time information visualization	4	4
4	Historic information visualization	2	2
5	Monitor real time position	5	6
6	Crash detection	8	9
7	Race parameter optimization	7	8
8	Ghost car	6	3
9	Autonomous car	9	7

Table 1: Scores given to technical implementation and to impact for each functionality

To decide whether to implement a functionality or not we developed a function that gives a total score based on the impact score and technical implementation score.

The function used is the following:

$$Total\ Score = \frac{Impact\ Score}{Technical\ Implementation\ Score} * k$$

Equation 1: How to calculate total score

The constant k is equal to 0.8 if the Technical Implementation Score is higher or equal to 7, if not k is equal to 1. This constant is set to adjust the technical implementation scores when they are too high, otherwise their weight would be low.

This function is developed in order to get the highest impact requiring a simpler implementation.

In Table 1 we can see the proposed improvements with their total scores. We take those which get a Total Score of 1 or more, in Table 1 they are colored green.

Nº	Name	Total Score
1	Monitor car metrics in real time	1,67
5	Monitor real time position	1,20
2	Driver registration	1,00
3	Real time information visualization	1,00
4	Historic information visualization	1,00
7	Race parameter optimization	0,91
6	Crash detection	0,90
9	Autonomous car	0,62
8	Ghost car	0,50

Table 2: Functionalities Total Score

Using this method, the project incorporates the improvements listed from 3.1.1 to 3.1.5.

First, the system allows the users to register. The names of the drivers that are in the race are displayed on a monitor.

During the race the acceleration of each car is obtained from the accelerometer. With it the system calculates the speed and the position of the cars. This information is recorded for future analysis and usage. The power given to the cars, the number of times each car changes lanes and the number of overtakes also is recorded. This specific information (throttle pressure, acceleration, speed, position and leaderboard) is to be shown on a display in real time to help drivers in their task of winning the race. All the information is to be collected for future analysis in the cloud.

After the race, the system will be able to analyze the data and give back information to the user. It will also be collected to be used in future races.

To summarize, the IoT project collects information, connects devices, processes data and shows it, providing a proper way to demonstrate how IoT works and how useful it can be.

Chapter 4. System developed

4.1. Elements of the system

The system built consist of both hardware and software. In addition to the software used in the final system there are other programs used to configure and deploy the devices involved. In this section general information about software, hardware, protocols, coding is provided.

This information will help understand the following sections as we will be referring to the items specified here.

4.1.1. Hardware

First, we are going to see the main hardware used to build the system: the control unit in the track, the gateway, the microcontroller and the development kits.

The **control unit** in the track is the one that sends the information in the rails, it is where the controllers are connected, where the configuration of the conditions for the races are defined. Here is where a serial cable can be connected to control certain parameters such as the speed of the cars or the fuel and to receive information such as the number of laps each car has done.

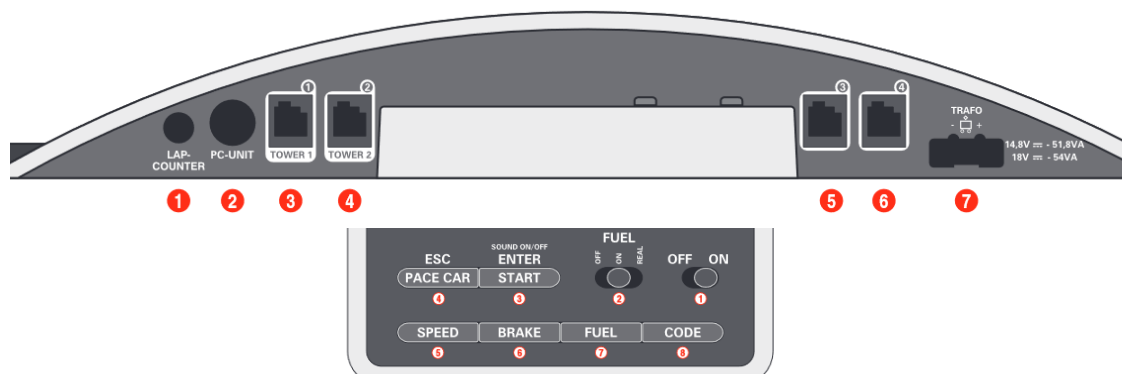


Figure 14: Control Unit connections and control elements

The racetrack used for the demonstrator is a Carrera Digital 132 track. It is a digital track which means it does not only provide power to the cars by means of a potentiometer, but it also can control many cars in the same rails. This means the cars can change lanes, the user can change the maximum speed of the cars and also, we can get the signal transmitted through the rails and know what is going on in the track.

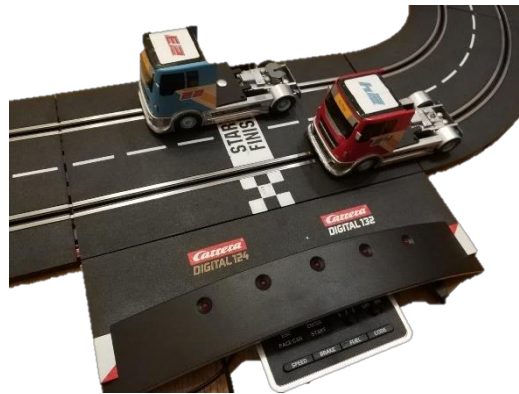


Figure 15: Racetrack control unit and trucks

The cars that are in the races are actually two trucks to make it easier to attach to them other components. These trucks have the system installed to decipher the signal of the track and they are powered by the current from the rails of the track.

The **gateway** is a device that allows our system to connect the different components and computes part of the data. Here is where the flows are run. These flows are what define what the system does and how. The gateway can

The **microcontroller**, which is an Arduino nano 33 IoT, is the one that has to get the information from the rails in the track. This information is not provided by the serial connected to the control unit. The data is collected and sent in a JSON file to the cloud from where it is retrieved to use to display the throttles of the cars.

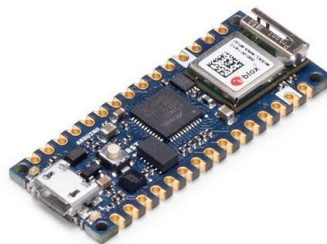


Figure 16: Microcontroller (Arduino Nano 33 IoT)

The **development kit** is small box, shown in Figure 17, that has many sensors inside and can send and receive the information via Wi-Fi and Bluetooth. The device used in this project is an XDK 110 Cross-Domain Development Kit made by Bosch. The sensors used for this project are the accelerometer and the gyroscope. The information from these sensors is sent through MQTT protocol each 100ms. We get 6 parameters and the timestamp each time the signal is sent. These six parameters are: acceleration in three axes and gyro in three axes. This information once sent is processed to get the speed

and position of each car. These boxes are set on top of each car, in fact they are not cars but trucks where it is easier to put them.



Figure 17: XDK-110

4.1.2. Software

For this project the main software for the working demonstrator is Altair SmartWorks, but other software is used: Node-RED, Arduino IDE, XDK-Workbench, Putty and MQTTBox.

- **SmartWorks** is the platform where the data is collected, stored and shown.
- **Node-RED** is a programming tool that provides a visual way to wire hardware and online services in a browser-based editor. The way the user works with it is by using nodes in a flow system. It has to be run on a computer or a gateway.
- **Arduino IDE** (Integrated Development Environment) is an open source program used to write and upload the code to a microcontroller.
- **XDK-Workbench** is the tool provided by Bosch that enables the development of software to be applied to the XDK 110.
- **PuTTY** is an open source terminal emulator that supports different networks protocols including SSH that is the one used to connect remotely to the gateway.
- **MQTTBox** is a program that helps to test the MQTT protocol in a system. It enables the creation of publishers and subscribers with a high degree of customization.

4.1.3. Protocols and other useful concepts

Throughout this memory there will be specific terms which might need to be clarified.

There are various protocols, code languages, file types and other concepts that are defined in this section.

- **MQTT** (Message Queue Telemetry Transport) is a lightweight connectivity protocol. The protocol is based on the principle of publishing messages and subscribing to topics, or "pub/sub". Multiple clients connect to a broker and

subscribe to topics that they are interested in. Clients also connect to the broker and publish messages to topics. Many clients may subscribe to the same topics and do with the information as they please. “The broker and MQTT act as a simple, common interface for everything to connect to” [13].

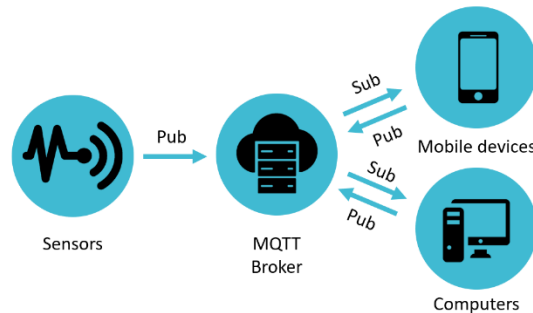


Figure 18: MQTT example

The four main concepts to keep in mind are:

- Broker acts a server connecting the devices.
- Topic acts as the recipient of messages, like a folder.
- Publish: is putting a message in a certain topic.
- Subscribe: is reading all messages that get to a topic.

QoS (Quality of service) defines the agreement between the publisher and the subscriber regarding the message delivered. There are three levels of QoS.

- If the QoS level is 0 the message sent will not be stored nor redelivered by the sender and will not be acknowledged by the receiver.
 - If the QoS level is 1 the message will be delivered at least once to the subscriber, but it can be delivered more than once.
 - If the QoS level is 2 The message will be delivered strictly one time accompanied by multiple acknowledgment messages confirming the delivery of the message.
- **JSON** (JavaScript Object Notation) is a lightweight data-interchange format. It is made to be easily understood by humans and computers alike.
 - **SSH** (Secure SHell) is a protocol for secure remote login and other secure network services over an insecure network.
 - **WoT** (Web of Things) intends to simplify the interoperability across different platforms. Some standards have been defined by the W3C (World Wide Web Consortium) to achieve this objective. Among them one of the most important is thing definition, which defines and describes an object connected to the web.

- **Thing**, we will be referring to the different parts of the IoT system as “things”. This is the way it is defined in the WoT.

4.2. System instrumentation and obtaining the signal

There are various devices that must be used to get the parameters in the demonstrator. We divide the instrumentation based on the signals that are processed in the system. These signals are the accelerometer signal, the gyroscope signal, the throttles signal and the signal transmitted and received by the control unit of the digital track.

4.2.1. Accelerometer

The accelerometer on the cars is in the XDK development kit explained in section 4.1.1. There is one XDK on each car.

The signal is read each 100ms and it is sent in a JSON file wirelessly (Wi-Fi) through MQTT. The code used in the XDK to obtain the acceleration signal has been modified from the code made by Solace IoT Team [14]. This code deployed in the XDK is stored in the internal memory, but there are some parameters that can be changed using a micro SD card. The parameters that are modifiable using the SD card are the ones related to the communications.

This signal is then processed to obtain the acceleration and the speed of the car. This is shown in section 4.3.1.

4.2.2. Gyroscope

As with the accelerometer, the gyroscope on the cars is in the XDK development kit explained in section 4.1.1.

The signal also is read each 100ms and it is sent in a JSON file wirelessly (Wi-Fi) through MQTT. The code used in the XDK to obtain the gyro signal has been modified from the code made by Solace IoT Team [14]. This code deployed in the XDK is stored in the internal memory, but there are some parameters that can be changed using a micro SD card. The parameters that are modifiable using the SD card are the ones related to the communications.

This signal is then processed to obtain the how much the car has turned, being able to determine that the car has made a complete lap. This is shown in section 4.3.2.

4.2.3. Throttles

Digital racetracks use a two-conductor system like analog systems; the main difference is that in analog systems the voltage between the conductors is modified to regulate the speed and in digital systems voltage is constant. The regulation of the speed in digital systems is then made with data packets transmitted interrupting the voltage in a certain

rhythm; with this system many cars can be connected to the same two conductors and still have different speeds.

The data packets consist in 10 different data words that are repeated continuously. Each data word is sent each 7.5ms; therefore, data packets are sent each 75ms. These data words are binaries that have between 8 and 13 bits. To make the 1s and 0s bits in each data word the track goes from around 13.9 Volts to 0. The data words are serial data words in Manchester code [15]; Manchester encoding uses a combination of the data signal with the signal of a clock, this way it is easier to know when the data transmission begins.

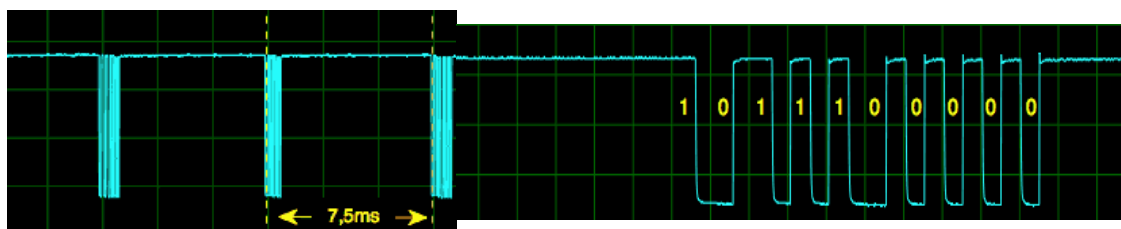


Figure 19: Racetrack signal [15]

To get the signal transmitted through the rails we use a microcontroller. The microcontroller has a limit in the input voltage that it can withstand so it requires a voltage divider. We can use two resistors and a diode to get the voltage divider.

The maximum voltage that there is between the rails is 14 Volts, as the microcontroller has a limited input of 3 Volts a voltage divider is applied. Using a breadboard, the divider is made with four 10kΩ resistors and a LED. The LED is used not only to reduce the voltage (1,6V) and limit currents, but also to know if it is correctly connected to the track. When the LED is on the circuit is correctly connected.

In Figure 20 a schema of the connection is shown, the signal enters the microcontroller through the digital PIN 2.

The resistors are distributed as shown in Figure 20 following the Equation 2 where R is the resistor used.

$$\frac{V_{out}}{(V_{in} - V_{LED})} = \frac{R}{R + 3R} = \frac{3}{12}$$

Equation 2: Voltage divider

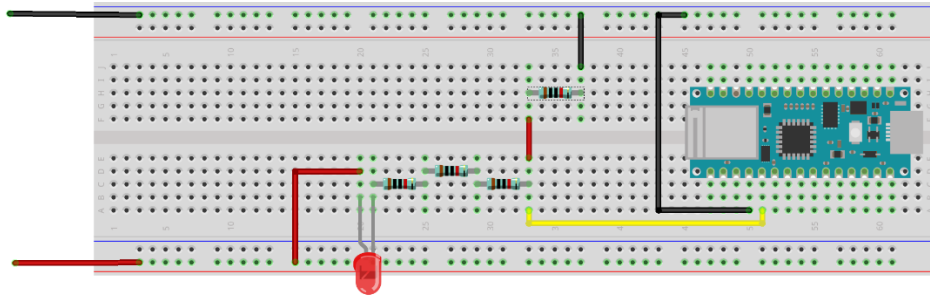


Figure 20: Arduino Circuit Sketch

Once the signal is within limits it is processed. The program used needs to detect when one of these signals is going to start. To do so we need to manage interruptions in the signal. Once we know when the signal has started to be transmitted, we begin to record the groups of ones and zeros in different words. To detect the word the microcontroller works with interruptions. The basic code to detect the interruptions and set the group of bits into words has been obtained from Peter Niehues code [16]. Then the code has been modified to send these words to be processed. This code is available in ANNEX II

These words are sent in a JSON file publishing them through MQTT. Then we take the words that give the information of the controllers and they are divided to select the part of them that tell the pressure in the throttles and if a change of lane is requested. This information is then sent to display. More information about how it is done is presented in section 0.

4.2.4. Digital track control unit

The track control unit powers the cars and controls the different signal within the track. These signals are the throttle to the cars, the lap count, the start of the race and other controls such as the maximum speed. This controls can be modified using the buttons on the control unit or using a serial connection, both shown in Figure 14 (serial connection uses the PC-UNIT connector).

For this project the connection to the control unit is made using serial cable. The cable connects the control unit to the gateway. The gateway contains and runs Python script that enables getting information from and sending commands to the control unit.

The code is modified from the one obtained from Thomas Kemmer in github [17]. This code has been modified to allow the remote use with MQTT. In ANNEX III are specified the changes.

The code is stored in various scripts that have to be run before the system is started. To do this we have to access the gateway, it can be done remotely using SSH protocol. As the gateway runs Linux we use the command:

```
nohup python -m carreralib /dev/ttyUSB0 &
```

This code starts the module called carreralib containing all the scripts necessary for the communication with the control unit using a USB connection. These scripts contain the parameters to make the necessary connections using MQTT.

4.3. Signal processing

The signals are processed in the gateway using Node-RED. Node-RED is a flow system that runs inside the gateway. The flows are build using different types of nodes; there are nodes for communication through MQTT and function nodes. The function nodes are the ones that allow us to make calculations to change the measurements from the devices into information that can be used for analysis and to display.

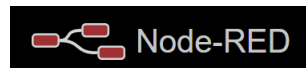


Figure 21: Node-RED logo

A basic and general flow for all the signals is:

- A node that subscribes to a topic from which it takes the signal input using MQTT.
- A node that converts between JSON string and JavaScript object.
- A function node that allows to select the data that we use and that calculates the output information.
- A node that publishes to a topic the information obtained.

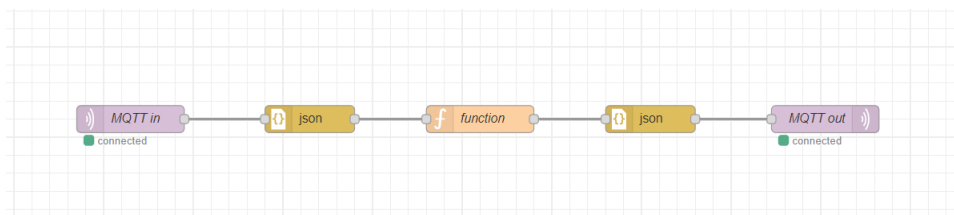


Figure 22: Basic Node-RED flow applied to all signals

4.3.1. Accelerometer

The information of the acceleration is received each 100ms. The device (XDK 110) collects and send the acceleration in three axes (x,y,z) but we ignore z axis because, as the track is horizontal, the acceleration read from this axis is always equal to the gravity acceleration.

Then with this information de speed and distance travelled is calculated. To calculate speed we should integrate the acceleration, but as we have discrete values of the acceleration we use an approximation. To do this we use Equation 3.

$$v = v_0 + \frac{1}{2} * (a + a_0) * (t - t_0)$$

Equation 3: Speed calculation from acceleration

In Equation 3 v stands for speed, a for acceleration and t for time. The $(t-t_0)$ used is the time from one message to the next, so we use 100ms.

The algorithm to calculate the distance traveled from the speed used is a similar to the one to calculate speed. We should integrate speed but we calculate the approximation shown in Equation 4.

$$x = x_0 + \frac{1}{2} * (v + v_0) * (t - t_0)$$

Equation 4: Position calculation

In Equation 4 x stands for position, v for speed, and t for time. The $(t-t_0)$ used is the time from one message to the next, so we use 100ms.

These calculations are not totally precise as errors are accumulated throughout the time.

4.3.2. Gyroscope

The signal received from the gyroscope is the angular speed in three axes. It can be processed to obtain the direction to which the device points. To do this a simple, yet not very precise, calculation can be made. This calculation is shown in Equation 5.

$$\theta = \theta_0 + \omega * (t - t_0)$$

Equation 5: Angle calculation

Where θ is the angle, ω is the angular speed and $(t-t_0)$ is the sampling time, which in this case is the time interval between receptions of the signals from the development kit (100ms).

4.3.3. Throttles decoding

The information received from the microcontroller is ciphered in a group of bit words, there are 10 words per message. Each message is sent through MQTT and then the gateway subscribes to the broker and topic where the information is being published. The information received is used to get the meaning. The meaning from these words has been extracted using the information provided by Stephan Heß [15].

The information taken from the track is received coded with the words in the following order:

- Programming data word.
- Pace and Ghost Car data word
- Active data word

- Controller data word 0
- Controller data word 4
- Controller data word 1
- Controller data word 5
- Controller data word 2
- Active data word
- Controller data word 3

Programming data word defines the maximum speed, brake and other parameters of the cars. **Active data word** defines if a controller is used or not. **Pace and Ghost Car data word** defines the speed from a car that drives at constant speed, this is used for expansions of the Carrera circuit that allow to use a safety car that will use this data. In this project Pace and Ghost Car data Word is not used. **Controller data word n**, being n a number from 0 to 5, defines the speed and if a car is going to change lane.

In Table 3 is shown the order of the words, their meaning, the number of bits in each word and a code for the meaning of each bit, which meaning is in Table 4.

Word	Meaning	Bit												
		1	2	3	4	5	6	7	8	9	10	11	12	13
1	Programming Data Word	1	V0	V1	V2	V3	P0	P1	P2	P3	P4	CA0	CA1	CA2
2	Pace and Ghost car Data Word	1	1	1	1									
3	Active Data Word	1	C0	C1	C2	C3	C4	C5	IE					
4	Controller data word 0	1	0	0	0	CL	S3	S2	S1	S0	GL			
5	Controller data word 4	1	1	0	0	CL	S3	S2	S1	S0	GL			
6	Controller data word 1	1	0	0	1	CL	S3	S2	S1	S0	GL			
7	Controller data word 5	1	1	0	1	CL	S3	S2	S1	S0	GL			
8	Controller data word 2	1	0	1	0	CL	S3	S2	S1	S0	GL			
9	Active data word	1	C0	C1	C2	C3	C4	C5	IE					
10	Controller data word 3	1	0	1	1	CL	S3	S2	S1	S0	GL			

Table 3: Throttle parameters

Name	Meaning
CL	Change Lane button (0 when pressed)
S	Speed
GL	Gasoline Level (1 when activated)
C	Controller
IE	One controller button pressed
CA	Controller Address
P	Parameter
V	Value

Table 4: Meaning of throttle parameters

The flow in Node-RED does the following: first it transforms the data sent in a JSON file into a JavaScript object; then functions are applied to take the bit words that have the information required for the project, from them we take the bits that have the information of the throttle pressure and the change of lanes. This information is then sent through MQTT to panopticon to display.

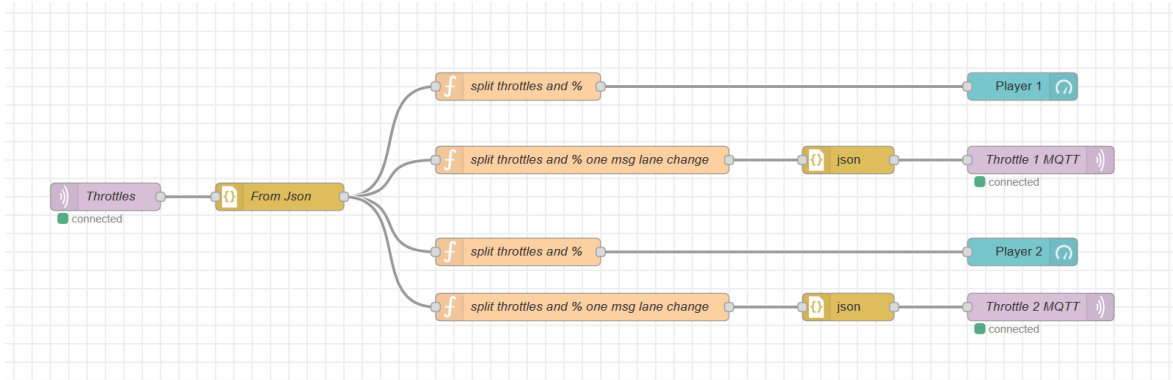


Figure 23: Throttle flow

The code for the flow shown in Figure 23 that processes the throttle data is in ANNEX IV.

4.3.4. Digital track control unit

The digital track control unit receives and emits data through the cable connected to the gateway. The signal processed is the one received from the carreralib that is sent using the MQTT protocol.

The data sent from the control unit are whether a car has passed through the finish line and the lap time. This information is processed in node-red to send it to the platform to be displayed.

The control unit receives information by subscribing to a topic where commands can be published. These commands allow the user to define the maximum speed of the cars and when the race starts.

4.4. System communication

The communication of the system is made using the MQTT protocol.

There are four main systems that need to be communicated. The microcontroller, the development kit, the track control unit and the gateway. All these systems are connected to the cloud.

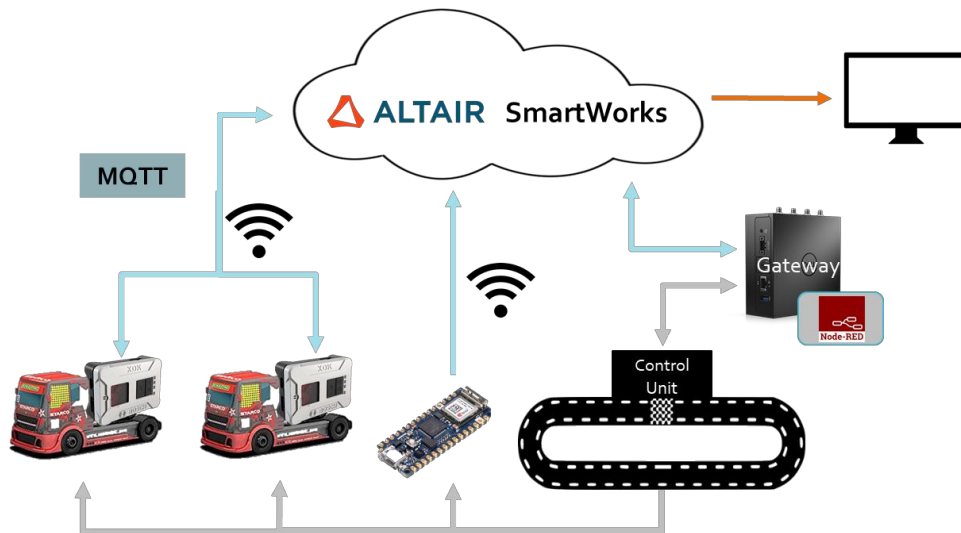


Figure 24: System communications architecture

4.4.1. Development Kit

The development kit is connected to the rest of the system using Wi-Fi connection and MQTT protocol. The main information for this communication is stored in a JSON file in a SD card inside the device. This information contains the name and password of the Wi-Fi network, the broker to connect to and the topics to subscribe and publish to.

The XDK (Cross Development Kit) is connected to a Wi-Fi and uses MQTT to send and receive messages. These messages are received and sent by the gateway using a subscription to the topic to which the XDK publishes and publishing to the topic to which the XDK is subscribed. Finally, this information is sent to the cloud to be displayed.



Figure 25: Development kit communication

4.4.2. Arduino

The communication between the Arduino and the rest of the system is made using the built-in Wi-Fi module. The microcontroller publishes the information to a topic to which the gateway is subscribed. The gateway processes the signal and publishes it again to a topic from which the cloud service takes it to display.

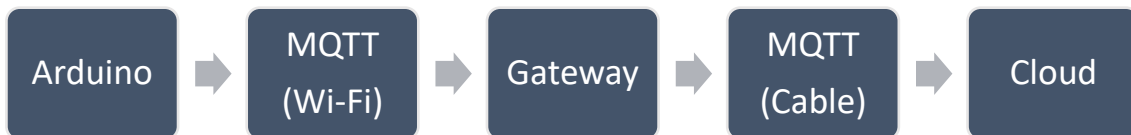


Figure 26: Arduino communication

4.4.3. Digital track Control Unit

The track Control Unit is connected to the system using a serial port. This connection is made using a cable that connects the control Unit to the gateway. The gateway sends the signals received from the control unit using MQTT. These signals are processed in the gateway and sent again with MQTT to be displayed.

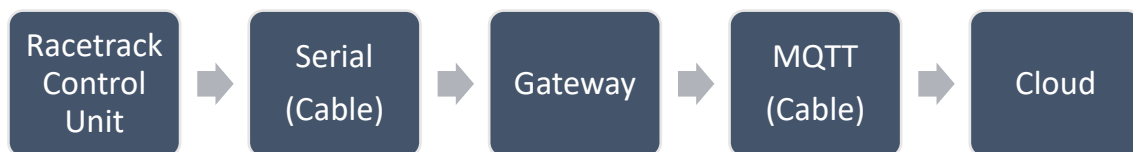


Figure 27: Track Control communication

The Gateway has two main focuses in the communication of the control unit. The first one is the obtention, interpretation and delivery of the signal sent by the control unit. This signal could be then used by someone that subscribes to the topic to which the gateway is publishing. The second one is the further processing of the signal to obtain information that is easier to visualize.

4.5. Design the backend Application

The backend application receives the data, works with it, stores the important data and displays the required information.

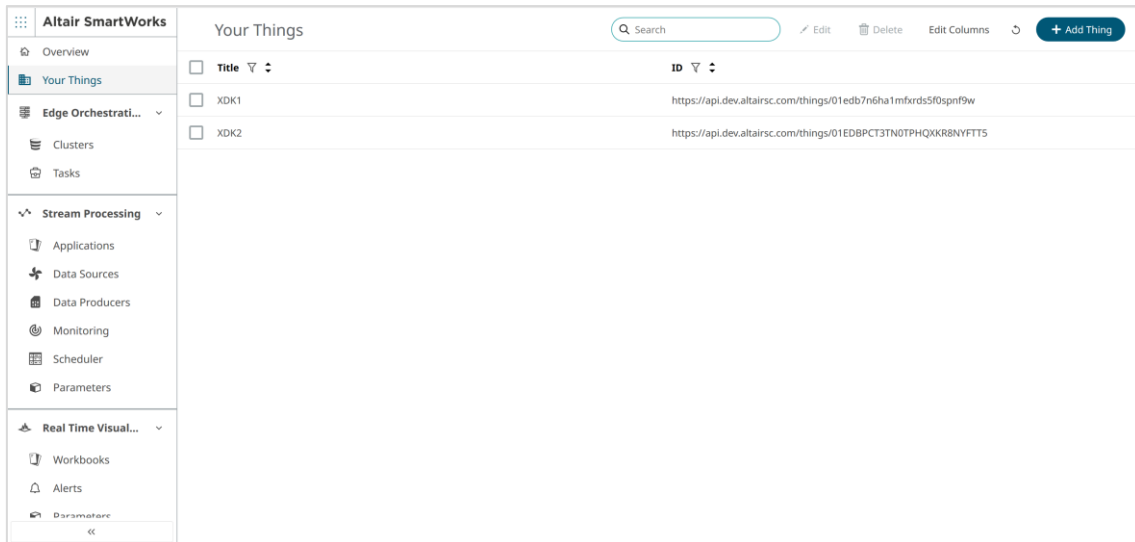


Figure 28: SmartWorks visualization of devices connected

The data is received by the backend application when the devices publish to the topic to which it is subscribed. This data is stored to be available for future analysis. An example of how it is stored is shown in Figure 29, Figure 30 and

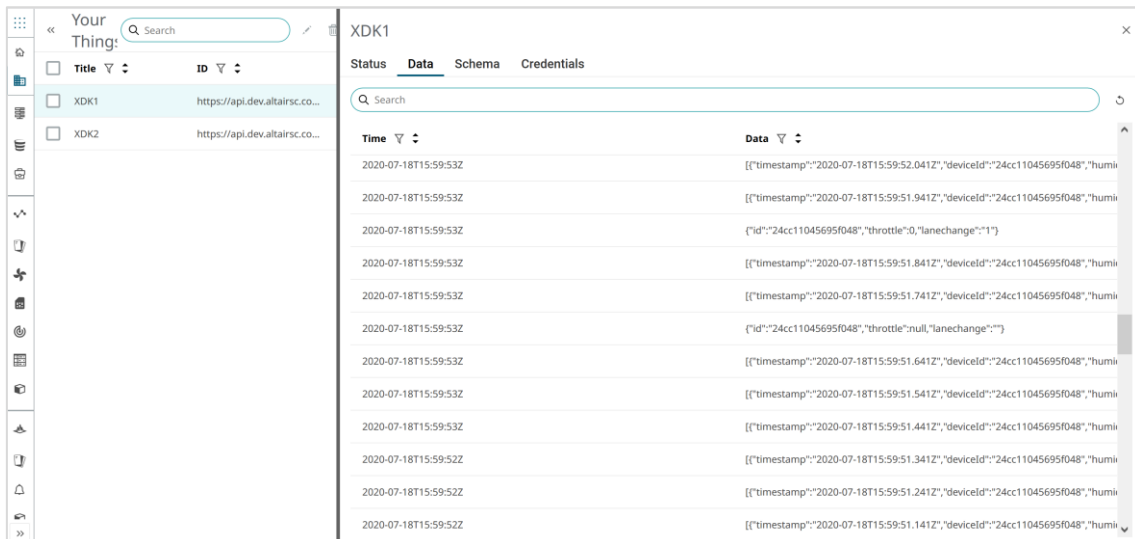


Figure 29: Data stored in SmartWorks



Figure 30: Detail of throttle data stored in SmartWorks

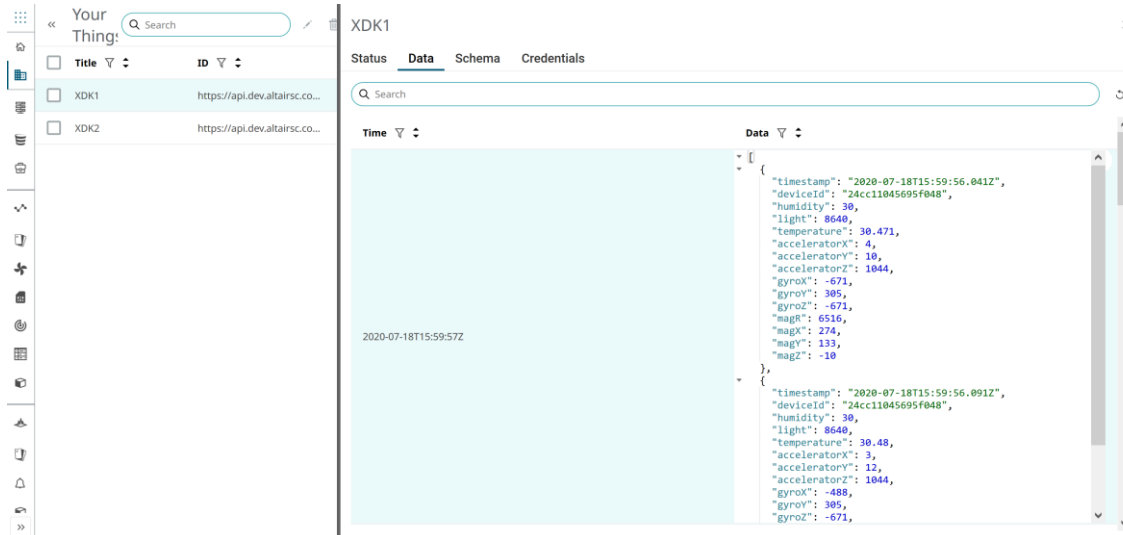


Figure 31: Detail of XDK stored data in SmartWorks

The data is worked with to obtain information about the system and to apply different responses to different sets of information.

The new information generated when working with the data is stored to be used in other applications.

The data and the information generated from it is further processed to be displayed. This information displayed is either the information that is being obtained in real time or the information that has been previously stored.

The main information shown is data about the driver and parameters obtained during the race (speed, lap times, leaderboards).

When using the demonstrator there will be several steps:

First when it is started the system ask the users to sign up using an email a name a surname and a nickname, this also creates an account in SmartWorks where they can see the information of the race.

Then when all the drivers are registered, we need to set the parameters of the race. First the type of race:

- it can be a race against the clock when there is only one driver then we have to select the time available to drive or the amount of laps that are going to be done, then we have to select if we want a ghost car on to compare our times with in real time; if we decide we want this functionality then we need to select the ghost car either from a previous time we have been racing and our time saved or the optimum one.
- It can be a normal race where the winner is the one who makes a number of laps the first; we have to select the number of laps that it will take to win
- It can be a race where the winner is the one that makes the fastest lap of a certain amount of time that has to be selected
- If there are going to be penalizations if the car goes out of the tracks
- If they want to see how they should be driving (accelerating or breaking)

The drivers in the race will be notified on a screen which car is the one they are going to drive and the controller to use.

Once the race type has been selected the lights will go from red to green and the race starts. During the race the information from the XDK and Arduino will be sent to the cloud every 10ms, in the cloud they will be processed and displayed in a screen where the drivers can see a leaderboard, the position of their cars in the track, number of laps, best lap time of each driver...

At the end of the race, being it when the first car finishes the number of laps predefined or when the time is up, the winner is displayed on the screen, and the leaderboard. At this instant is when an email is sent to all the participants with the leaderboard, the lap times they have done and other information.

4.6. Budget

This project does not aim to get a direct money benefit from building the demonstrator, but it aims to gain clients for the platform of the collaborating entity. Therefore, as there is no possible way to calculate the economic viability, a budget of the project is presented.

Component	Components used	Units	Price/Unit	Total
Gateway	Dell Edge Gateway 3001	1	420.00 €	420.00 €
Master track	Carrera Digital 132	1	274.32 €	274.32 €
Development Kit	XDK 110	2	236.50 €	473.00 €
Router POE Injector	Netgear ProSafe GS108P	1	84.99 €	84.99 €
Wi-Fi Router	Wi-Fi Router	1	30.00 €	30.00 €
Microcontroller with Wi-Fi	Arduino Nano 33 IoT	1	25.20 €	25.20 €
Breadboard and electronic components	Breadboard and electronic components	1	10.00 €	10.00 €
USB MiniDin6	USB MiniDin6	1	6.99 €	6.99 €
Ethernet cable	Ethernet cable	1	1.00 €	1.00 €
			Total	1,325.50 €

Table 5: Hardware budget

All the hardware needed to build the demonstrator sums 1325.50€. To this amount the price of the software used and the staff salary should be added. As many programs are free (Arduino IDE, XDK Workbench) the only cost that has to be taken into account is the price of SmartWorks.

Chapter 5. Results analysis

When all the devices were programmed and connected the whole system was trialed to see that the goals were satisfied. This involved checking all the connections were working during the time the demonstrator was being used.

To check the connections, we had to try the different capacities of the elements connected. We had to see that the signals sent to the control unit were being received and that the tasks commanded were followed. We had to check that the data sent by the different devices was being received in the platform so it could be processed to obtain more information.

The best way to certify that the system was working as expected was to see if the information was shown correctly in a display. This information displayed had to be shown in real time to certify the signals were sent without unnecessary delays. This visualization can be seen in Figure 32.



Figure 32: Real time visualization of measurements

The visualization in Figure 32 shows the accelerations obtained from the sensors mounted on the vehicles, the throttle pressure being made by the drivers and a lap count that also displays the lap times.

Figure 32 was taken shortly after the start of the race. In it we can see that the drivers are pressing the throttles to a mid-point. In the graphics on the left we can see the variation of the acceleration that affects the speeds, when the acceleration is positive the speed increases and when its negative the speed decreases.

Chapter 6. Conclusions and future projects

The usage of IoT for many disciplines is going to experience an important rise in the next years. Having the tools to deploy it with ease will help developers. The company that gets there first with a better technology will win a great market share. In order to show customers how IoT can be deployed many examples and demos will be built. In this project one of these demonstrators has been built using a variety of technologies such as wired and wireless, MQTT protocol and local and cloud processing.

Although the project is made with a slot car racetrack, which seems like a toy, it is a perfect example of how the IoT can be deployed in a system. This is because this project shows the main capabilities of IoT, allowing to get data from many sources in many places, process all this data both in a gateway and the cloud and use this information for different purposes such as visualization or analysis.

In this project different devices have been used to get the data. These devices range from affordable and easily programable microcontrollers to more complex and expensive devices; this shows that the IoT is applicable to nearly any device we can think of, making it useful to apply to almost any discipline.

Using standards enables the interconnection of devices conceived to use different languages. For instance, the MQTT protocol enables the transmission of data between devices with different configurations; the usage of JSON files makes it easier for the devices involved in the connection and the user to understand the messages. All these technologies make IoT universally applicable.

As the system is connected using the internet the devices involved in it can be placed anywhere where a connection to the web can be attained. This makes it useful for big and dispersed systems.

For future projects there are many ways to go on; one possible project to be developed could be to improve the data analysis collected from the IoT racetrack; another one could be reducing costs of the whole system and make the devices used more compact; it also could be to implement more of the suggested functionalities to the system.

Bibliography

- [1] Altair Engineering, "www.altair.com," [Online]. Available: <https://www.altair.com/about/>. [Accessed 07 04 2020].
- [2] B. Butler, "www.networkworld.com," 11 02 2013. [Online]. Available: <https://www.networkworld.com/article/2163430/paas-primer--what-is-platform-as-a-service-and-why-does-it-matter-.html>. [Accessed 13 07 2020].
- [3] "www.altairsmartworks.com/smartcore-overview," [Online]. Available: <https://www.altairsmartworks.com/smartcore-overview>. [Accessed 13 07 2020].
- [4] T. Schubert, B. Völker, M. Pfeifer and B. Becker, "https://www.wb.uni-freiburg.de/inhalte/pdfs/oh-projekt/iems/the_smart_minifab_paper," 2018. [Online]. Available: https://www.wb.uni-freiburg.de/inhalte/pdfs/oh-projekt/iems/the_smart_minifab_paper. [Accessed 13 02 2020].
- [5] "petras-iot.org," [Online]. Available: <https://petras-iot.org/about-us/>. [Accessed 04 03 2020].
- [6] "www.bregroup.com," [Online]. Available: <https://www.bregroup.com/>. [Accessed 04 03 2020].
- [7] "petras-iot.org/update/iot-in-the-home-demonstrator/," [Online]. Available: <https://petras-iot.org/update/iot-in-the-home-demonstrator/>. [Accessed 04 03 2020].
- [8] S. Cela, B. Cabe and J. Vermillard, "wiki.eclipse.org/IoT/M2MIWG/Demonstrator," 04 02 2014. [Online]. Available: <https://wiki.eclipse.org/IoT/M2MIWG/Demonstrator>. [Accessed 13 02 2020].
- [9] Internte of Things for European Small and Medium Enterprises, "iot4smes.eu/en/demonstrators.aspx," [Online]. Available: <https://www.iot4smes.eu/en/demonstrators.aspx>. [Accessed 21 02 2020].
- [10] A. Medina Merino, "openaccess.uoc.edu/webapps/o2/bitstream/," 18 04 2018. [Online]. Available:

<http://openaccess.uoc.edu/webapps/o2/bitstream/10609/80645/4/artmedmerTFM0618memoria.pdf>. [Accessed 21 02 2020].

- [11] A. Mayer, "blog.qlick.com," 2 November 2017. [Online]. Available: <https://blog.qlik.com/the-qlik-iot-race-game>. [Accessed 12 February 2020].
- [12] "opcau.github.io/anki/IoTRaceTrack/," 2016. [Online]. Available: <https://opcau.github.io/anki/IoTRaceTrack/>. [Accessed 13 02 2020].
- [13] R. Light, "mosquitto.org," [Online]. Available: <http://mosquitto.org/man/mqtt-7.html>. [Accessed 14 07 2020].
- [14] Solace IoT Team, "github.com/solace-iot-team," [Online]. Available: <https://github.com/solace-iot-team/Bosch-IoT-Showcase>. [Accessed 25 05 2020].
- [15] S. Heß, "http://slotbaer.de," [Online]. Available: <http://slotbaer.de/carrera-digital-124-132/12-d132-d124-daten-protokoll.html>. [Accessed 24 05 2020].
- [16] P. Niehues, "www.wasserstoffe.de," [Online]. Available: <http://www.wasserstoffe.de/carrera-hacks/protocol-decode/index.html>. [Accessed 14 04 2020].
- [17] T. Kemmer, "github.com/tkem," [Online]. Available: <https://github.com/tkem/carreralib>. [Accessed 12 06 2020].
- [18] United Nations, "www.un.org," [Online]. Available: <https://www.un.org/sustainabledevelopment/sustainable-development-goals/>. [Accessed 19 07 2020].

ANNEXES

ANNEX I. SDG

According to the web of the United Nations:

The Sustainable Development Goals (SDGs) are the blueprint to achieve a better and more sustainable future for all. They address the global challenges we face, including those related to poverty, inequality, climate change, environmental degradation, peace and justice. The 17 Goals are all interconnected, and in order to leave no one behind, it is important that we achieve them all by 2030. [18]

For this annex we have looked the possible impacts of the project developed on the Sustainable Development Goals. As the system developed is a demonstrator of how a technology can be used, we will be taking into consideration the use that could be made with the technology (IoT) shown.

Following this consideration, the main impact this project has is on the SDG 9 (Industry, Innovation and Infrastructure), which intends to build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation.



Figure 33: SDG 9

Inside this goal there are some specific targets, some of which could be attained using what has been developed in this project. For instance, target 9.1 “Promote inclusive and sustainable industrialization” is one that is linked with this project. By making companies and industries more connected using IoT we could increase the number of manufactured products and still use the resources in the most efficient way.

IIoT is the term used for Industrial Internet of Things. This IIoT can be used to improve the sustainability of the industry, making the different steps in the production of a product the most efficiently possible.

IoT can also make the infrastructure more resilient by providing information of whether maintenance is needed, this is done processing obtained data measured from sensors set on the infrastructure to analyze.

IoT has another benefit when being implemented in developing countries for the industrialization as it can have a great impact on their economy at a low cost.

Another Sustainable Development Goal that could be impacted from the development of this project is goal 11 “Sustainable Cities and communities”. The target 11.2 aims to expand public transport. If public transport were to be made more appealing to the citizens, its usage would increase. To make users take public transport instead of private transport it should be more affordable, and users should not need to wait long times to go from one place to another. All of this could be tackled using a combination of data gathered using IoT with data analysis.



Figure 34: SDG 11

Target 11.b, which includes the resources efficiency in cities, could be reached using connecting different systems within a city or within larger regions. Smart cities, as they are known, use the computation of data gathered to improve the lives of the citizens. In these smart cities the IoT can be implemented to obtain the best ways to use in many ways such as illumination on the streets, to define better the garbage recollection timetable and to distribute traffic so there are no traffic jams, that are highly pollutant.

To conclude, IoT can help solve many problems and enterprises could understand from this project how it works and how it could be implemented to their new ideas.

ANNEX II. Arduino throttles code

```

/*
  WiFi + MQTT publisher + arduino tracks 2

  This example is written for a network using WPA encryption. For
  WEP or WPA, change the Wifi.begin() call accordingly.

  Circuit:
  * Analog inputs attached to pins A0 through A5 (optional)

  * This work is licensed under the Creative Commons Attribution-
  NonCommercial-ShareAlike 3.0 Unported License.
  * To view a copy of this license, visit
  http://creativecommons.org/licenses/by-nc-sa/3.0/.
  *
  * This license allows you to remix, tweak, and build upon my work
  non-commercially, as long as
  * you credit Peter Niehues and license the new creations under the
  identical terms.
  *
  created 8 april 2020
  by Fidel Sanz
  modified 14 April 2020
  by Fidel Sanz

  */

#include <SPI.h>
#include <WiFiNINA.h>
#include <PubSubClient.h>

//CAMBIAR DATOS MQTT

#include "arduino_secrets.h" //sensitive data in the Secret
tab/arduino_secrets.h
char ssid[] = SECRET_SSID; // your network SSID (name)
char pass[] = SECRET_PASS; // your network password (use for
WPA, or use as key for WEP)
int keyIndex = 0; // your network key Index number
(needed only for WEP)

const char* mqttServer =
"mqtt.dev.altairsc.com";//"mqtt.eclipse.org";//"mqtt.dev.altairsc.com"
;//"mqtt.eclipse.org"
char* mqttUsername = MQTT_USER;// "user";
char* mqttPassword = MQTT_PASS;//"12345678";
const char clientId[] = "";//"fsanz03::240"; // MQTT Client_ID

char pubTopic[] =
"589";//"topic/test";//"589";//"arduino/throttlesState";// "data/240";
//"arduino/throttlesState"

WiFiClient espClient;
PubSubClient client(espClient);

int status = WL_IDLE_STATUS;

```

```
//WiFiServer server(80);

void setup_wifi()
{
  // check for the WiFi module:
  if (WiFi.status() == WL_NO_MODULE) {
    Serial.println("Communication with WiFi module failed!");
    // don't continue
    while (true);
  }

  String fv = WiFi.firmwareVersion();
  if (fv < WIFI_FIRMWARE_LATEST_VERSION) {
    Serial.println("Please upgrade the firmware");
  }

  // attempt to connect to Wifi network:
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network. Change this line if using open or
    WEP network:
    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(10000);
  }
  //server.begin();
  // you're connected now, so print out the status:
  printWifiStatus();
  digitalWrite(LED_BUILTIN, HIGH);
}

void mqtt_connect() //MQTT
{
  // Loop until we're connected
  while (!client.connected())
  {
    status = WiFi.begin(ssid, pass);
    if (status != WL_CONNECTED){
      digitalWrite(LED_BUILTIN, LOW);
      setup_wifi();
    }
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect(clientId, mqttUsername, mqttPassword))
    {
      Serial.println("connected");
    } else
    {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 1 second");
      // Wait 1 second before retrying
      delay(1000);
    }
  }
}
}
```



```

const int dataPin = 2; // the number of
the pin to read carrera racetrack data
int wordCount = 0; // counter

boolean wordChange = false; // indicates a
new word is ready for further analysing

long currentWord = 0; // assemble Bits
to complete data words, used in interrupt routine
long Word = 0; // save data
word for further computing, used in main loop and interrupt
long Words[11]; // store 10
datawords in array with counts 1..10

unsigned long intervalMicros = 0; // will store
the time between two bit changes (microseconds)
unsigned long previousMicros = 0; // will store
last time state when data signal has changed (microseconds)
unsigned long currentMicros = 0; // will store
the current runtime (microseconds)

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(115200); // initialize
  serial bus
  pinMode(dataPin, INPUT); // initialize
  the dataPin as an input
  attachInterrupt(2, manchesterDecode, CHANGE); // whenever
  levels on dataPin change, start interrupt routine

  pinMode(LED_BUILTIN, OUTPUT);
  setup_wifi();
  client.setServer(mqttServer, 1883);
}

void loop() {

  if( wordChange == true ){ // only start
  when interrupt routine has assembled a complete word
  wordChange = false; // reset word
  change indicator
  if( Word > 4000 ) // synchronize
  to longest dataword with 13 bits
  wordCount = 1; // longest word
  is saved on first position in array
  Words[ wordCount ] = Word; // save the
  actual word in array
  wordCount++; // increase
  counter
  }

  if ( wordCount == 11 ){ // if all 10
  words are stored, write them to serial
  for (wordCount=1; wordCount < 11; wordCount++){ //
  Serial.print( wordCount, DEC ); // print number
  of dataword

```

```

        Serial.print( "\t" );                // print tab
        Serial.println( Words[ wordCount ], BIN );    //} print
dataword in binary format
    }

    String one = String(Words[1], BIN);
    String two = String(Words[2], BIN);
    String three = String(Words[3], BIN);
    String four = String(Words[4], BIN);
    String five = String(Words[5], BIN);
    String six = String(Words[6], BIN);
    String seven = String(Words[7], BIN);
    String eight = String(Words[8], BIN);
    String nine = String(Words[9], BIN);
    String ten = String(Words[10], BIN);
    String opening = "{";
    String closing = "}";
    String comma = ",";
    String program = "\"prog\": ";
    String ghost = "\"gho\": ";
    String active = "\"act\": ";
    String controlzero = "\"C0\": ";
    String controlone = "\"C1\": ";
    String controltwo = "\"C2\": ";
    String controlthree = "\"C3\": ";
    String controlfour = "\"C4\": ";
    String controlfive = "\"C5\": ";

    String json_message = opening +
        program + one + comma +
        ghost + two + comma +
        active + three + comma +
        controlzero + four + comma +
        controlfour + five + comma +
        controlone + six + comma +
        controlfive + seven + comma +
        controltwo + eight + comma +
        active + nine + comma +
        controlthree + ten + closing;

    char buf[200];
    json_message.toCharArray(buf, 200);
    client.publish(pubTopic, buf);

    Serial.println("-----");    //
    wordCount = 1;                // reset counter

    delay( 10 );                // wait two
seconds
}

if (!client.connected())
{
    mqtt_connect();
}
client.loop();
}
void printWifiStatus() {
    // print the SSID of the network you're attached to:

```

```

Serial.print("SSID: ");
Serial.println(WiFi.SSID());

// print your board's IP address:
IPAddress ip = WiFi.localIP();
Serial.print("IP Address: ");
Serial.println(ip);

// print the received signal strength:
long rssi = WiFi.RSSI();
Serial.print("signal strength (RSSI):");
Serial.print(rssi);
Serial.println(" dBm");
}

void manchesterDecode() { //////////////////////////////////////////////////
    currentMicros = micros(); // get current
runtime
    intervalMicros = currentMicros - previousMicros; // calculate
interval
    if (intervalMicros > 75 && intervalMicros < 125){// is full clock
pulse?
        previousMicros = currentMicros; // synchronise
        currentWord = currentWord << 1; // shift bits left
        if ( digitalRead( dataPin ) == LOW ) // is pin level
LOW?
            bitSet( currentWord,0 ); // received digital
1
        return; } // leave interrupt
        if ( intervalMicros > 6000 ) { // is word rate?
            Word = currentWord; // save bits for
main loop
            currentWord = 0; // reset bits
            bitSet( currentWord,0 ); // first bit is
always 1
            wordChange = true; // indicate a new
word
            previousMicros = currentMicros; // synchronise
            return; } // leave interrupt
}

```

“arduino_secrets.h” is a file which contains sensitive information; it has the following structure:

```

#define SECRET_SSID "AccespointSSID"
#define SECRET_PASS "Password"

#define MQTT_USER "fsanz05@fsanz05"
#define MQTT_PASS "Zfbz0xH8uqRJckWv"

```

ANNEX III. Carreralib modified code

The code in carreralib needs a file called x.py with the following code:

```
import paho.mqtt.client as mqtt
import time

def on_disconnect(client, userdata, rc):
    if rc !=0:
        print("Unexpected Disconnect")

def on_message(client, userdata, message):
    print("message received " ,str(message.payload.decode("utf-8")))
    print("message topic=",message.topic)

broker_address="10.145.8.165"
client = mqtt.Client("Track_Control")
client.on_disconnect = on_disconnect
client.on_message = on_message
client.connect(broker_address)

client.loop_start()
client.subscribe("track/control")

while True:
    a=1

client.loop_stop()
```

ANNEX IV. Throttle processing

In this section the code used to create the flows that processes the throttle data.

```
[
  {
    "id": "37bb4c26.0e3a7c",
    "type": "tab",
    "label": "Flow 4",
    "disabled": false,
    "info": ""
  },
  {
    "id": "3485dbdd.500114",
    "type": "ui_gauge",
    "z": "37bb4c26.0e3a7c",
    "name": "",
    "group": "64fcefef.13ad4",
    "order": 1,
    "width": 0,
    "height": 0,
    "gtype": "gage",
    "title": "Player 1",
    "label": "",
    "format": "{{value | number:1}}%",
    "min": 0,
    "max": "100",
    "colors": [
      "#00b500",
      "#e6e600",
      "#ca3838"
    ],
    "seg1": "",
    "seg2": "",
    "x": 1200,
    "y": 420,
    "wires": []
  },
  {
    "id": "f857dacd.f6de08",
    "type": "mqtt in",
    "z": "37bb4c26.0e3a7c",
    "name": "Throttles",
    "topic": "arduino/throttlesState",
    "qos": "2",
    "datatype": "auto",
    "broker": "ffba3def.23ac1",
    "x": 200,
    "y": 540,
    "wires": [
      [
        "b5036948.1be2b"
      ]
    ]
  },
  {
    "id": "b5036948.1be2b",
    "type": "json",

```

```

    "z": "37bb4c26.0e3a7c",
    "name": "From Json",
    "property": "payload",
    "action": "",
    "pretty": false,
    "x": 390,
    "y": 540,
    "wires": [
      [
        "cb692951.c18b1",
        "6cf5d9c2.a7a05",
        "fd8365ce.9d90a",
        "7d8f73bc.a49e14"
      ]
    ]
  },
  {
    "id": "bb6aea56.1c9208",
    "type": "ui_gauge",
    "z": "37bb4c26.0e3a7c",
    "name": "",
    "group": "64fcefef6.13ad4",
    "order": 2,
    "width": 0,
    "height": 0,
    "gtype": "gage",
    "title": "Player 2",
    "label": "",
    "format": "{{value | number:1}}%",
    "min": 0,
    "max": "100",
    "colors": [
      "#00b500",
      "#e6e600",
      "#ca3838"
    ],
    "seg1": "",
    "seg2": "",
    "x": 1200,
    "y": 600,
    "wires": []
  },
  {
    "id": "6cf5d9c2.a7a05",
    "type": "function",
    "z": "37bb4c26.0e3a7c",
    "name": "split throttles and %",
    "func": "\nvar x0=global.get('x0') || 0; //to retrieve a
variable \n\nvar c0 = String(msg.payload.C0);\nvar c0s =
c0.substring(5,9);\nvar c0sd = parseInt(c0s, 2);\nvar
sp=(c0sd)/15*100;\n\nvar msg1={payload: sp};\nvar msg2={payload:
x0};\n\nif (x0!=sp){\n  x0 = sp;\n  global.set('x0',x0);// to
store a variable\n  return [msg1];\n}\n\nreturn;\n",
    "outputs": 1,
    "noerr": 0,
    "x": 650,
    "y": 420,
    "wires": [
      [
        "3485dbdd.500114"
      ]
    ]
  }
}

```

```

    ]
  ],
  {
    "id": "cb692951.c18b1",
    "type": "function",
    "z": "37bb4c26.0e3a7c",
    "name": "split throttles and %",
    "func": "\nvar x1=global.get('x1') || 0; //to retrieve a
variable \n\nvar c0 = String(msg.payload.C1);\nvar c0s =
c0.substring(5,9);\nvar c0sd = parseInt(c0s, 2);\nvar
sp=(c0sd)/15*100;\n\nvar msg1={payload: sp};\nvar msg2={payload:
x1};\n\nif (x1!=sp){\n  x1 = sp;\n  global.set('x1',x1);// to
store a variable\n  return [msg1];\n}\n\nreturn;\n",
    "outputs": 1,
    "noerr": 0,
    "x": 650,
    "y": 600,
    "wires": [
      [
        "bb6aea56.1c9208"
      ]
    ]
  },
  {
    "id": "fd8365ce.9d90a",
    "type": "function",
    "z": "37bb4c26.0e3a7c",
    "name": "split throttles and % one msg lane change",
    "func": "\nvar xaa=global.get('xaa') || 0; //to retrieve a
variable \n\nvar c0 = String(msg.payload.C0);\nvar c0s =
c0.substring(5,9);\nvar c0sd = parseInt(c0s, 2);\nvar
sp=(c0sd)/15*100;\n\nvar msg1;\nvar msg1a={};\nvar msg2a={};\n\nvar
cl0aa=global.get('cl0aa') || 0; //to retrieve a variable \n\nvar ch0 =
c0.substring(4,5);\n\nmsg1a.id=\"24cc11045695f048\"\n\nmsg1a.throttle
=sp;\n\nmsg1a.lanechange =ch0;\n\nif (xaa!=sp){\n  xaa = sp;\n
global.set('xaa',xaa);// to store a variable\n  \n  msg1={payload:
msg1a};\n}\n\nif (cl0aa!=ch0){\n  cl0aa=ch0;\n
global.set('cl0aa',cl0aa);// to store a variable\n\n  msg1={payload:
msg1a};\n}\n\nreturn [msg1];",
    "outputs": 1,
    "noerr": 0,
    "x": 720,
    "y": 500,
    "wires": [
      [
        "a98040d0.07181"
      ]
    ]
  },
  {
    "id": "a98040d0.07181",
    "type": "json",
    "z": "37bb4c26.0e3a7c",
    "name": "",
    "property": "payload",
    "action": "",
    "pretty": false,
    "x": 1010,
    "y": 500,
  }
}

```

```

    "wires": [
      [
        "92e25942.4f98b"
      ]
    ],
    {
      "id": "92e25942.4f98b",
      "type": "mqtt out",
      "z": "37bb4c26.0e3a7c",
      "name": "Throttle 1 MQTT",
      "topic": "fsanz05/data/01edb7n6halmfxrds5f0spnf9w",
      "qos": "0",
      "retain": "false",
      "broker": "b2f85550.575388",
      "x": 1220,
      "y": 500,
      "wires": []
    },
    {
      "id": "7d8f73bc.a49e14",
      "type": "function",
      "z": "37bb4c26.0e3a7c",
      "name": "split throttles and % one msg lane change",
      "func": "\nvar xaa2=global.get('xaa2') || 0; //to retrieve a
variable \n\nvar c1 = String(msg.payload.C1);\nvar cls =
c1.substring(5,9);\nvar clsd = parseInt(cls, 2);\nvar
sp=(clsd)/15*100;\nvar msg1;\nvar msg1a={};\nvar msg2a={};\n\nvar
cl0aa2=global.get('cl0aa2') || 0; //to retrieve a variable \n\nvar ch0
= c1.substring(4,5);\n\nmsg1a.id=\"242a95065695cb19\"\n\nmsg1a.throttle
=sp;\nmsg1a.lanechange =ch0;\n\nif (xaa2!=sp){\n  xaa2 = sp;\n
global.set('xaa2',xaa2);// to store a variable\n  \n
msg1={payload: msg1a};\n}\n\nif (cl0aa2!=ch0){\n  cl0aa2=ch0;\n
global.set('cl0aa2',cl0aa2);// to store a variable\n\n
msg1={payload: msg1a};\n}\n\nreturn [msg1];",
      "outputs": 1,
      "noerr": 0,
      "x": 720,
      "y": 660,
      "wires": [
        [
          "4a303b02.c76244"
        ]
      ]
    },
    {
      "id": "4a303b02.c76244",
      "type": "json",
      "z": "37bb4c26.0e3a7c",
      "name": "",
      "property": "payload",
      "action": "",
      "pretty": false,
      "x": 1010,
      "y": 660,
      "wires": [
        [
          "df3af1f8.358f08"
        ]
      ]
    }
  ]

```



```
},
{
  "id": "df3af1f8.358f08",
  "type": "mqtt out",
  "z": "37bb4c26.0e3a7c",
  "name": "Throttle 2 MQTT",
  "topic": "fsanz05/data/01EDBPCT3TN0TPHQXKR8NYFTT5",
  "qos": "0",
  "retain": "false",
  "broker": "535d5ae7.7917d4",
  "x": 1220,
  "y": 660,
  "wires": []
},
{
  "id": "64fcefe6.13ad4",
  "type": "ui_group",
  "z": "",
  "name": "Throttles",
  "tab": "76571d2e.aa083c",
  "order": 1,
  "disp": true,
  "width": "5",
  "collapse": false
},
{
  "id": "ffba3def.23ac1",
  "type": "mqtt-broker",
  "z": "",
  "name": "",
  "broker": "mqtt.eclipse.org",
  "port": "1883",
  "clientid": "",
  "usetls": false,
  "compatmode": false,
  "keepalive": "60",
  "cleansession": true,
  "birthTopic": "arduino/throttlesState",
  "birthQos": "0",
  "birthPayload": "",
  "closeTopic": "",
  "closeQos": "0",
  "closePayload": "",
  "willTopic": "",
  "willQos": "0",
  "willPayload": ""
},
{
  "id": "b2f85550.575388",
  "type": "mqtt-broker",
  "z": "",
  "name": "SmartWorks XDK 1",
  "broker": "mqtt.dev.altairsc.com",
  "port": "1883",
  "clientid": "",
  "usetls": false,
  "compatmode": false,
  "keepalive": "60",
  "cleansession": true,
  "birthTopic": "",
```

```
    "birthQos": "0",  
    "birthPayload": "",  
    "closeTopic": "",  
    "closeQos": "0",  
    "closePayload": "",  
    "willTopic": "",  
    "willQos": "0",  
    "willPayload": ""  
  },  
  {  
    "id": "535d5ae7.7917d4",  
    "type": "mqtt-broker",  
    "z": "",  
    "name": "SmartWorks XDK 2",  
    "broker": "mqtt.dev.altairsc.com",  
    "port": "1883",  
    "clientid": "",  
    "usetls": false,  
    "compatmode": false,  
    "keepalive": "60",  
    "cleansession": true,  
    "birthTopic": "",  
    "birthQos": "0",  
    "birthPayload": "",  
    "closeTopic": "",  
    "closeQos": "0",  
    "closePayload": "",  
    "willTopic": "",  
    "willQos": "0",  
    "willPayload": ""  
  },  
  {  
    "id": "76571d2e.aa083c",  
    "type": "ui_tab",  
    "z": "",  
    "name": "Throttles",  
    "icon": "dashboard",  
    "order": 2,  
    "disabled": false,  
    "hidden": false  
  }  
]  
]
```

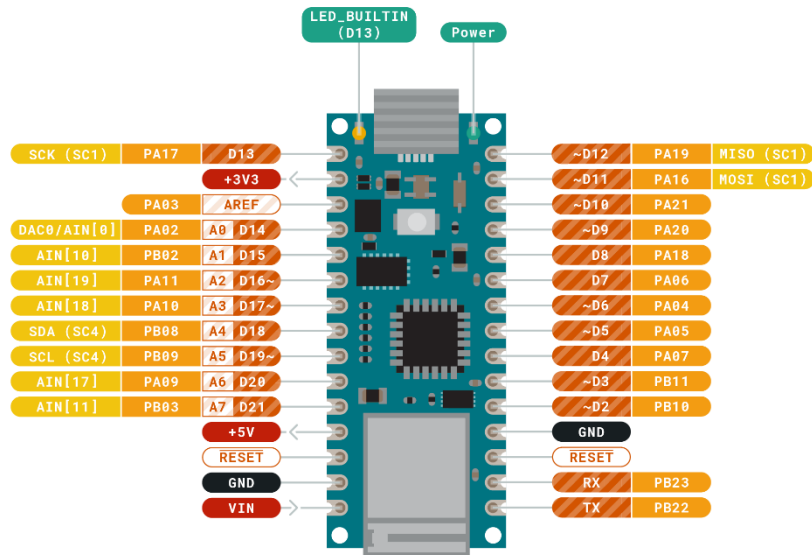
ANNEX V. Arduino Nano 33 IoT datasheet

Microcontroller	SAMD21 Cortex®-M0+ 32bit low power ARM MCU (datasheet)
Radio module	u-blox NINA-W102 (datasheet)
Secure Element	ATECC608A (datasheet)
Operating Voltage	3.3V
Input Voltage (limit)	21V
DC Current per I/O Pin	7 mA
Clock Speed	48MHz
CPU Flash Memory	256KB
SRAM	32KB
EEPROM	none
Digital Input / Output Pins	14
PWM Pins	11 (2, 3, 5, 6, 9, 10, 11, 12, 16 / A2, 17 / A3, 19 / A5)
UART	1
SPI	1
I2C	1
Analog Input Pins	8 (ADC 8/10/12 bit)
Analog Output Pins	1 (DAC 10 bit)
External Interrupts	All digital pins (all analog pins can also be used as interrupt pins, but will have duplicated interrupt numbers)
LED_BUILTIN	13
USB	Native in the SAMD21 Processor
IMU	LSM6DS3 (datasheet)
Length	45 mm
Width	18 mm
Weight	5 gr (with headers)




ARDUINO

NANO 33 IoT



Ground	Internal Pin	Digital Pin	Microcontroller's Port
Power	SWD Pin	Analog Pin	
LED	Other Pin	Default	

ARDUINO.CC



This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

ANNEX VI. Bosch XDK 110 Datasheet

Bosch Connected Devices and Solutions GmbH | Datasheet

Cross Domain Development Kit | XDK

Start your Sensor X - perience



The universal programmable sensor device & prototyping platform for any IoT use case you can imagine!

Make use of the power to monitor, control and analyze your product remotely over Bluetooth or Wireless Network. In this way, devices, products or machines become connective and smarter. XDK is now released for 24/7 usage giving you the freedom to use it either for short-term proof of concepts or long-term projects. Inclusive of multiple Micro-Electromechanical Systems (MEMS) sensors, various parameters for condition monitoring or predictive maintenance get recorded.

You can decide between a rapid prototyping kit (XDK110) and as a professional bundle (XDK Node) in a package of 10 pieces.

The XDK110 device was designed for rapid prototyping and allows users an easy transition from prototype to mass production by providing a clear road to product development.

The XDK Node – Professional Bundle includes 10 pieces of XDK110 devices with an optimized scope of delivery which enables a cost effective deployment for larger projects and simplifies the installation.

BUILT-IN SENSORS



Accelerometer



Acoustic sensor



Digital light sensor



Gyroscope



Humidity sensor



Magnetometer



Pressure sensor



Temperature sensor

APPLICATION ADVANTAGES

- ▶ All-in-one sensor kit: no need for component selection, hardware assembly, or deployment of a real-time operating system
- ▶ Algorithm library
- ▶ Example code in open source licensing
- ▶ Drivers for all system components included
- ▶ Secure data protocol
- ▶ Small form factor (Length 60 mm x Width 40 mm x Height 22 mm; Weight 54 g)
- ▶ Built-in lithium ion rechargeable battery
- ▶ Functional extensibility via the included extension board
- ▶ High-level API for the standard user and low-level API for the power user
- ▶ PC and MAC based development tools for Windows, LINUX and MacOS make it an easy to work with tool for any developer
- ▶ CE, FCC, IC, IMDA, ACMA, NTC and NBTC certified | further on request



INCLUDED IN DELIVERY

XDK110 - Rapid Prototyping Kit

- ▶ XDK Development Kit
- ▶ "XDK Gateway" extension board for easy access to additional MCU functionality
- ▶ 10 cm connector cable
- ▶ Micro USB 2.0 connector cable
- ▶ Mounting plate and screws

XDK Node – Professional Bundle

- ▶ Bundle of 10 XDK110 devices (without "XDK Gateway" and without 10 cm connector cable)
- ▶ Micro USB 2.0 connector cable
- ▶ Mounting plate and screws

MAIN COMPONENTS

- ▶ Bluetooth 4.0 low energy IEEE 802.15.1
- ▶ Wireless LAN IEEE 802.11b/g/n
- ▶ 32-Bit microcontroller (ARM Cortex M3), 1MB Flash, 128 kB RAM
- ▶ Internal Li-Ion rechargeable battery 560 mAh
- ▶ Integrated antennas



BOSCH
Invented for life

OPERATING CONDITIONS

- ▶ Indoor use
- ▶ Operating temperature range -20 °C ... 60 °C, (0 °C ... 45 °C for battery charging)
- ▶ Storage temperature range -20 °C ... 60 °C
- ▶ Humidity range 10...90 %rH (non-condensing)
- ▶ IP Rating IP 30 (IEC 60529)
- ▶ Supply Voltage 5 V DC

MEASUREMENT RANGES

- ▶ Accelerometer $\pm 2 \dots \pm 16$ g (programmable)
- ▶ Gyroscope ± 125 °/s ... ± 2000 °/s (programmable)
- ▶ Magnetic field strength ± 1300 μ T (X,Y-Axis); ± 2500 μ T (Z-Axis)
- ▶ Light sensor 0.045 lux ... 188,000 lux ; 22-bit
- ▶ Temperature -20 °C ... 60 °C [limited by XDK operating conditions]
- ▶ Pressure 300...1100 hPa
- ▶ Humidity 10...90 %rH (non-condensing)
[limited by XDK operating conditions]

SAMPLING RATE

- ▶ Accelerometer BMA280 2000 Hz
- ▶ Gyroscope BMG160 2000 Hz
- ▶ Magnetometer BMM150 300 Hz
- ▶ Hum./Press./Temp. BME280 182 Hz
- ▶ Inertial Measurement Unit 1600 Hz (Accelerometer);
BMI160 3200 Hz (Gyroscope)

SOFTWARE

Free software download for XDK110 & XDK Node from the website (<https://xdk.io/software-downloads>)

- ▶ Integrated development environment supplied with XDK Workbench (Eclipse)
- ▶ LWM2M communication protocol
- ▶ Extensive libraries and modular source code enable the developer to fully understand the system
- ▶ Datagram Transport Layer Security (DTLS)

USER INTERFACE

- ▶ Power switch
- ▶ Green system LED to display the state of charging
- ▶ 3 programmable status LEDs (red, orange, yellow)
- ▶ 2 programmable push-buttons
- ▶ Micro SD card slot
- ▶ Interface for J-Link Debug-probe
- ▶ Interface for extension board

GET IN CONTACT WITH US

Bosch Connected Devices and Solutions
E-Mail: support@bosch-connectivity.com

VISIT OUR WEBSITE

www.xdk.io



Technical data subject to modification without notice.

© Bosch Connected Devices and Solutions GmbH | 2017. All rights reserved, also regarding and disposal, exploitation, reproduction, editing, distribution, as well as in the event of application for industrial property rights. January, 10, 2018