Instituto Tecnológico de Estudios Superiores de Monterrey
Campus Monterrey

REPORTE DE REQUERIMIENTOS Y DE DISEÑO

Diseño de Sistemas Embebidos Avanzados

TE2004B

Arturo José Murra López A01236090
José Alfredo García Alvarado A01275161
Oscar Cárdenas Gómez A01284083
Ricardo Gonzalo Cruz A01284147

Monterrey, 1 de Diciembre de 2022

Product Vision Board

Vision: Create a system which performs optimally by taking into account the environmental variables around it further optimizing its life expectancy and diminishing its need for maintenance.

Target: John Deere, farmers, tractor owners and manufacturers. Those who mainly use the vehicle and undertake its maintenance.

Needs: A system that can take environmental variables for optimizing it's performance and tractor life expectancy.

Product: A system that can change to an optimal tyre pressure depending on the terrain the tractor is being used on diminishing the stress caused on the tyres.

Business Goals: Become engineering leaders that sell durable and efficient products that will withhold the test of time. Make products that present less flaws diminishing their need for maintenance.

Development and Team Work

Backlog Scrum Methodology

Leyenda

- **Epic**
    - **User Story**
        - **Task**

- Design Epic
    - Understand the problem
        - Read The Problem
        - Research Possible solutions
    - Identify needed Materials
        - Research needed materials
        - Check for access to materials
        - Get Materials
    - Identify approach of program logic
        - Review the already gathered information
        - Analyze what you need to solve and what is not necessary
    - Design microcontroller system
        - Take the approach and begin attacking the problem through it.
        - Start dividing and conquering to make problem easier

        - Take our knowledge to create the circuit tht fixes the already identified problems
- Independent Development and Testing Epic
    - Build microcontroller system
        - Analyze the design of the system
        - Build the system based on the previous design of it
    - Review microcontroller system
        - Analyze what is already designed
        - Check for any possible mistakes
    - Fix microcontroller system
        - In case the system does not work, go back to the building and designing of system
        - If it works move on
    - Build Prototype
        - If there were no mistakes in the review, begin build
        - Double check you built everything just as designed
    - Test Prototype
        - Revise and make sure the prototype works
        - If it works, move on
    - Fix Prototype
        - If there were mistakes in the prototype testing, fix the prototype and go back to the design and building of this.
- Joint Development and Testing Epic
    - Mount microcontroller on Prototype
        - Test all functions separately first
        - Integrate functions step by step
    - Test Prototype with microcontroller
        - Make a test on Lab Conditions
        - Make a test on Field Conditions
    - Fix Prototype with microcontroller
        - View flaws and errors presented on tests
        - Detect possible solutions to amend those mistakes
        - Implement new ideas to fix the errors
        - Test again (see epic before this one)
- Present final result of our System
    - Write documentation
        - Check for the requirements of the documentation
        - Gather the information that is required
        - Create a technical report with the previously gathered information,
    - Esthetic care of final prototype
        - Eliminate building imperfections
        - Align final product with a color pattern
        - Take careness of overall presentation

Scrum Methodology

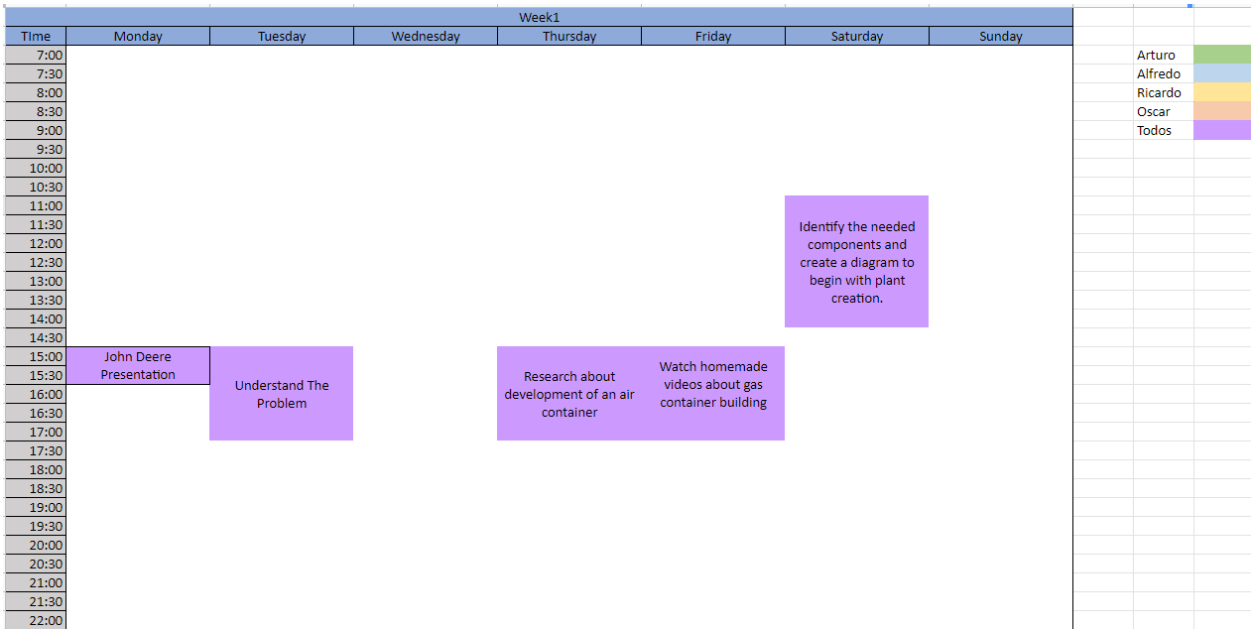


Figure 1. Scrum week 1

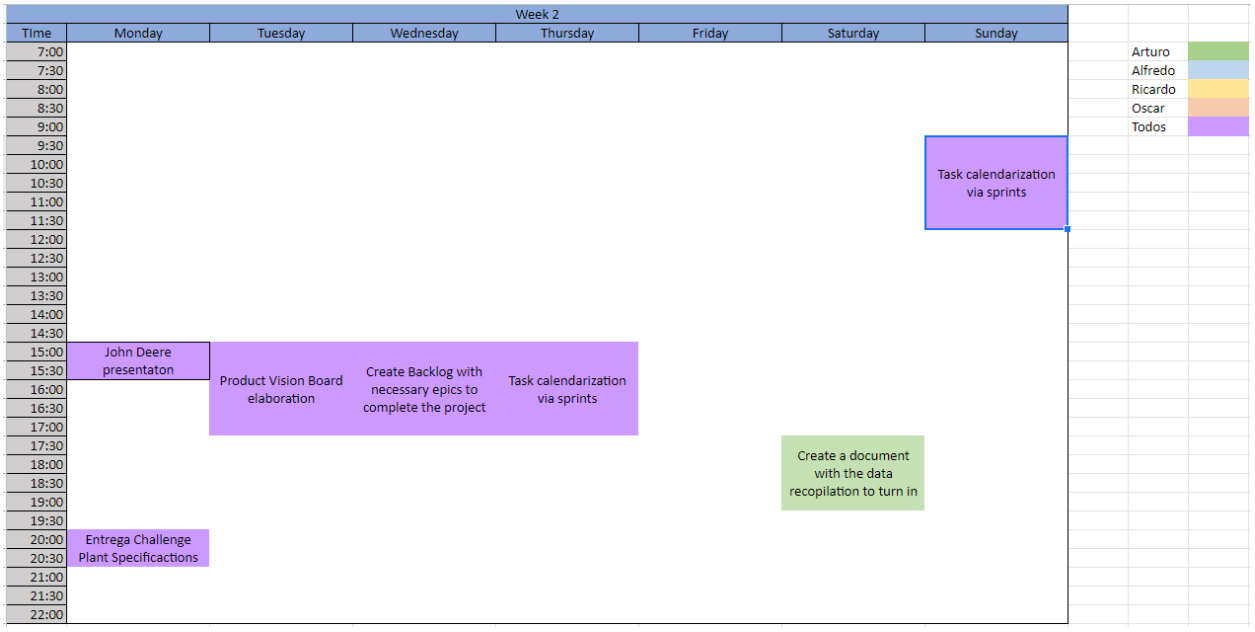


Figure 2. Scrum week 2

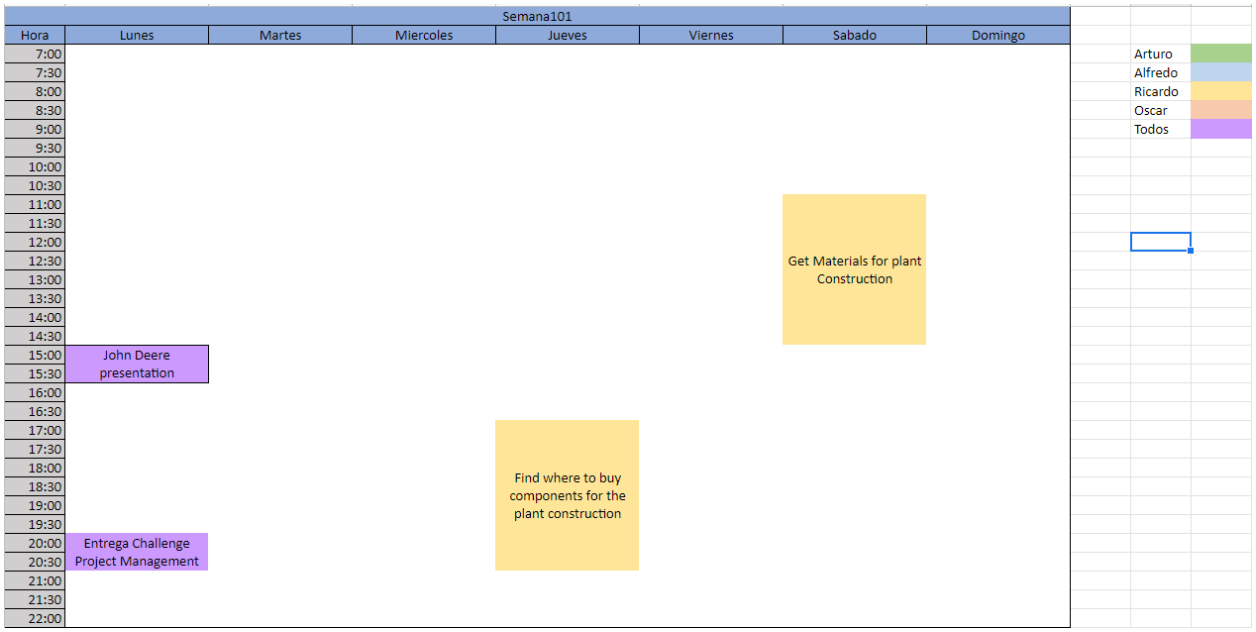


Figure 3. Scrum week 3

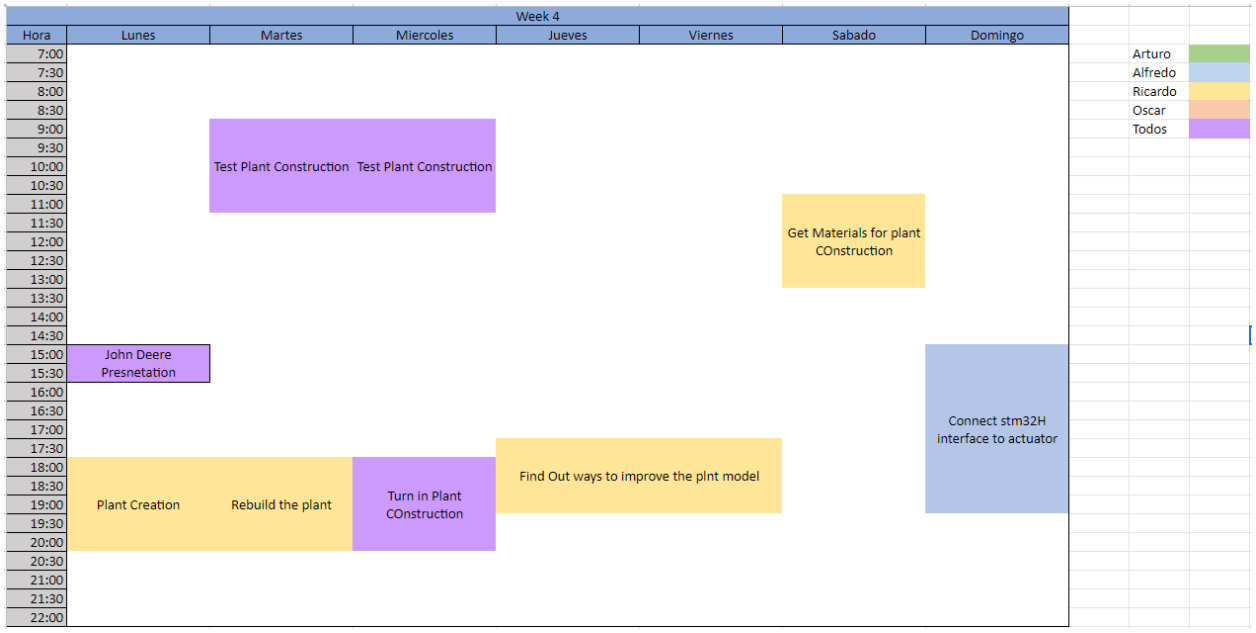


Figure 4. Scrum week 4

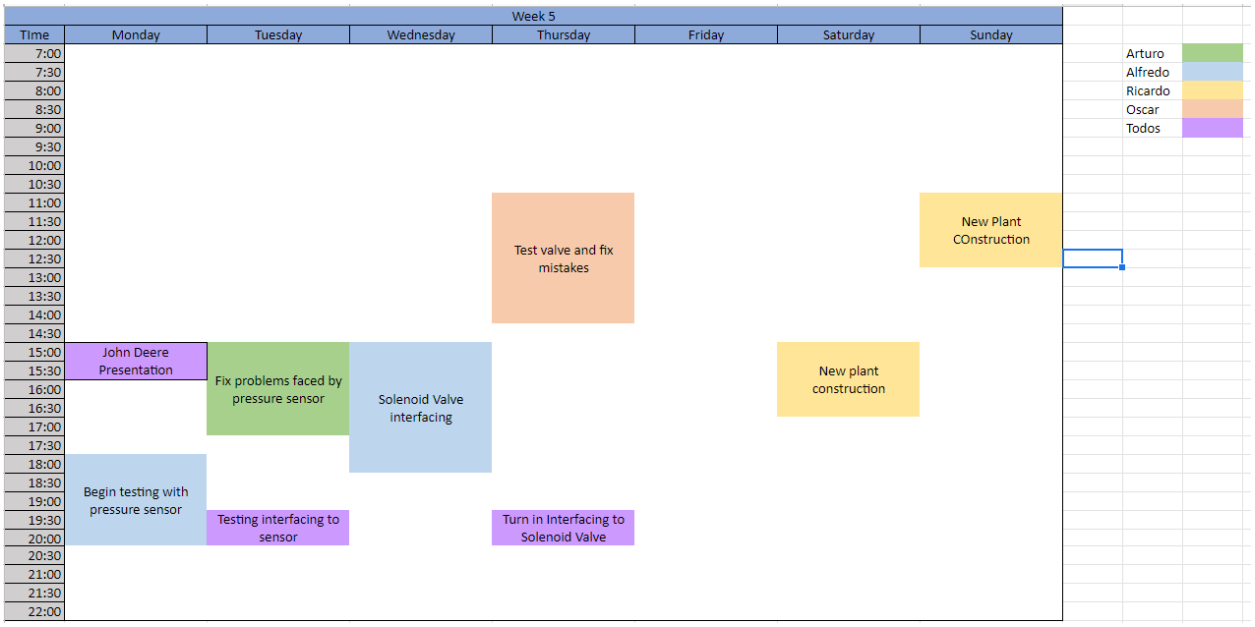


Figure 5. Scrum week 5

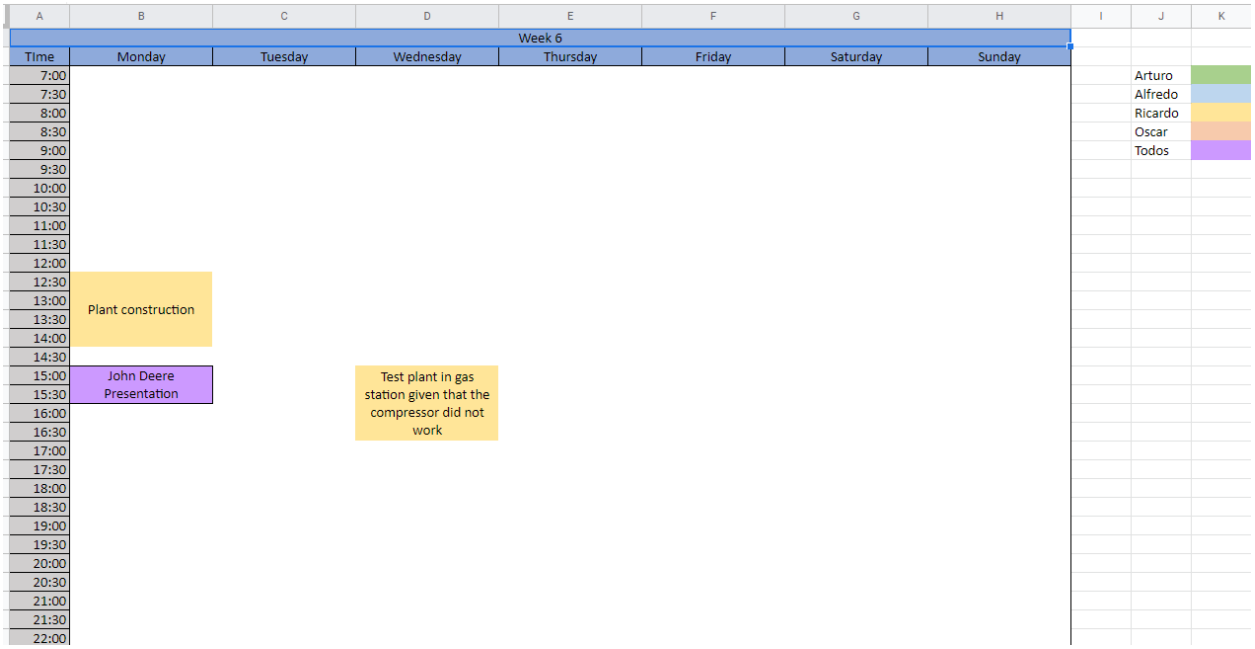


Figure 6. Scrum week 6

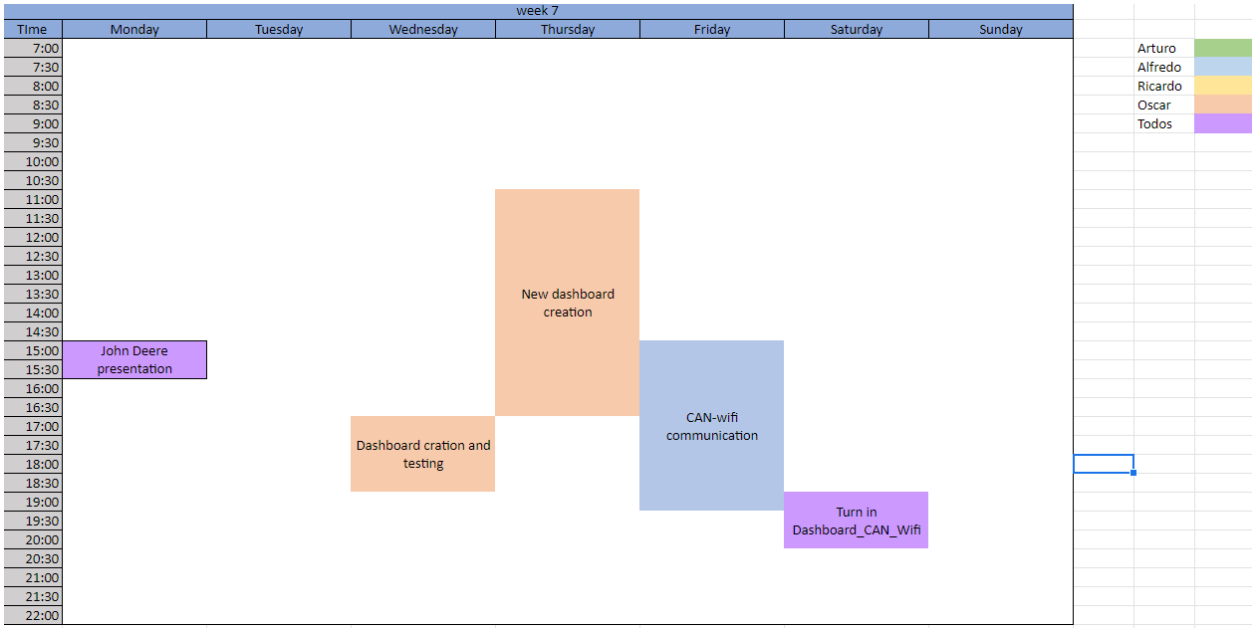


Figure 7. Scrum week 7

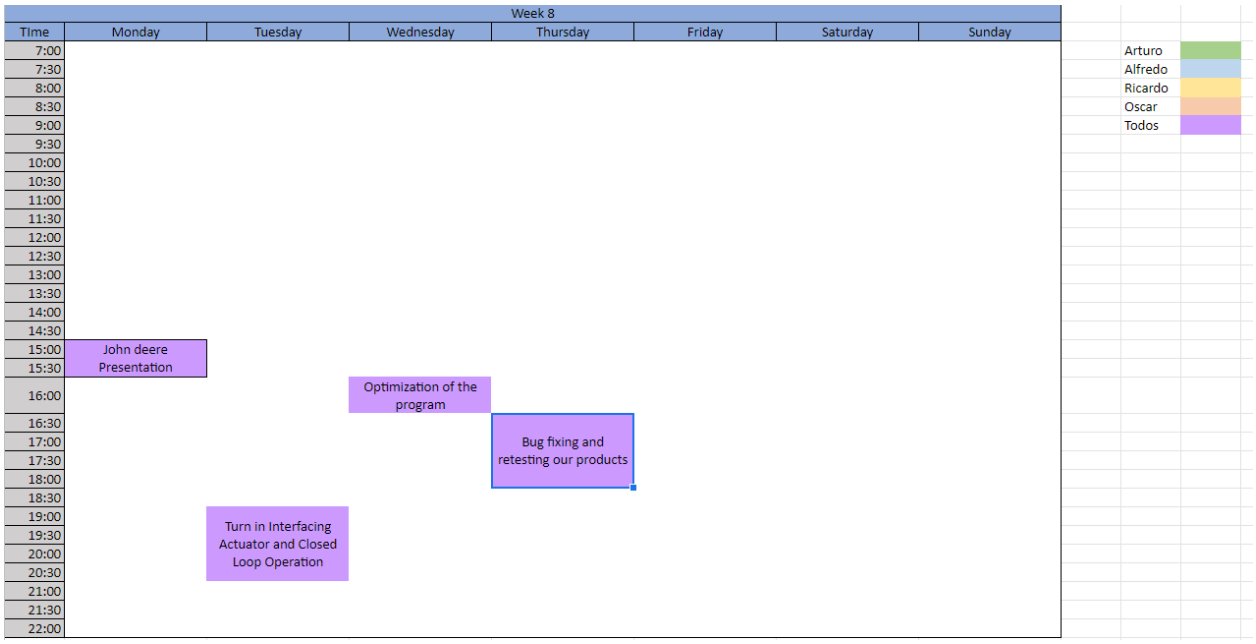


Figure 8. Scrum week 8

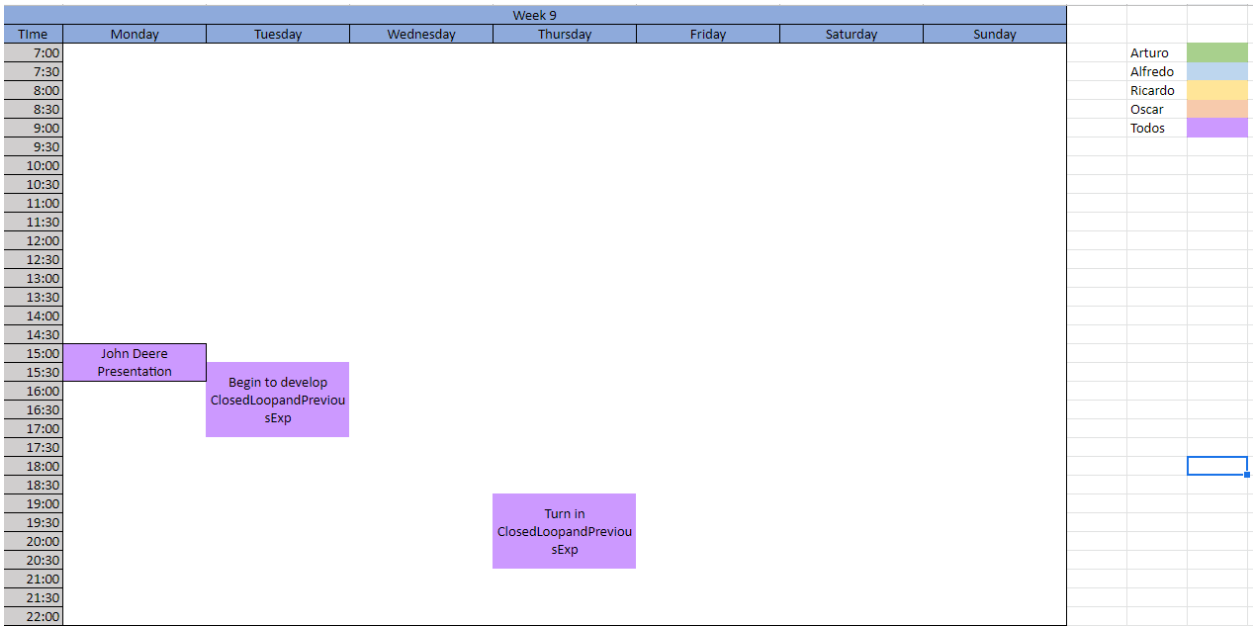


Figure 9. Scrum week 9

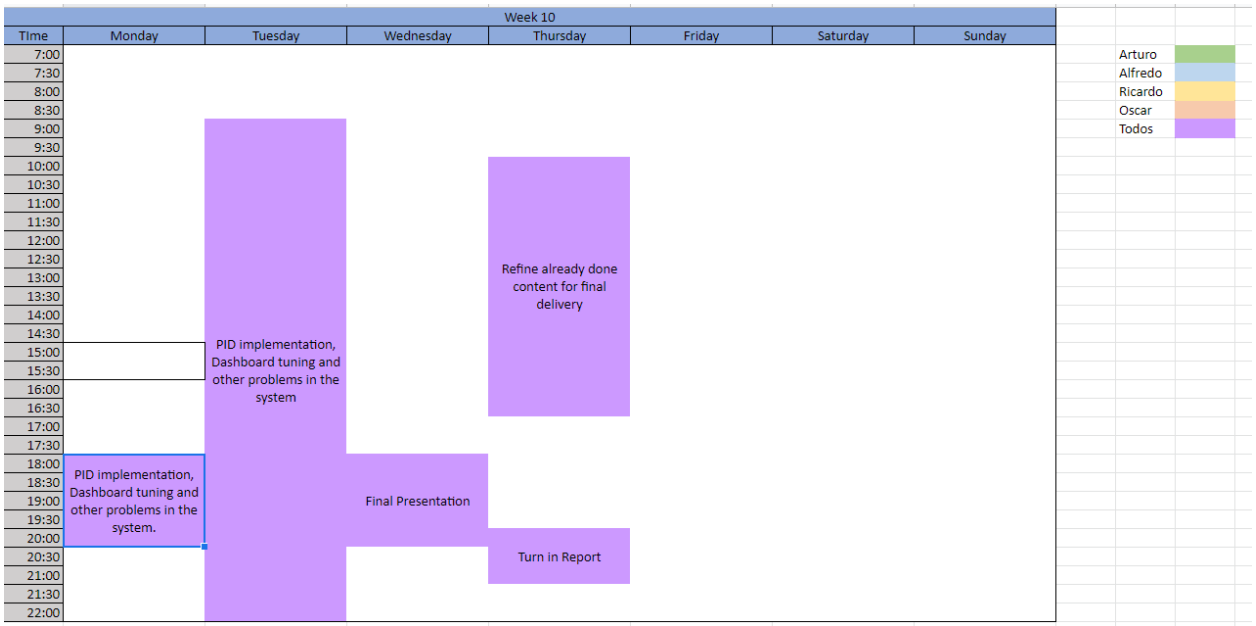


Figure 10. Scrum week 10

Distribution and Workflow

We distributed each of the tasks between our team members in accordance to each preference and expertise. The advantage of our team is that all of our members have previous experience with using some of the elements that are present in our project. So

when it came to assigning tasks each one of us had already in mind the steps needed to follow and create to complete the project.

Updates to our Workflow and Distribution Dynamic

One thing that as a team considers that is of importance for this type of projects is that we could have as an upgrade a better in depth knowledge of the topics that the project requires before entering full into the project. The reason why is that with beforehand knowledge we could optimize our workflow and distribution into even considering parallel working methods where we can all continue making progress without having to stop and wait for the other team member to finish a part of the project.

UML Context Diagram

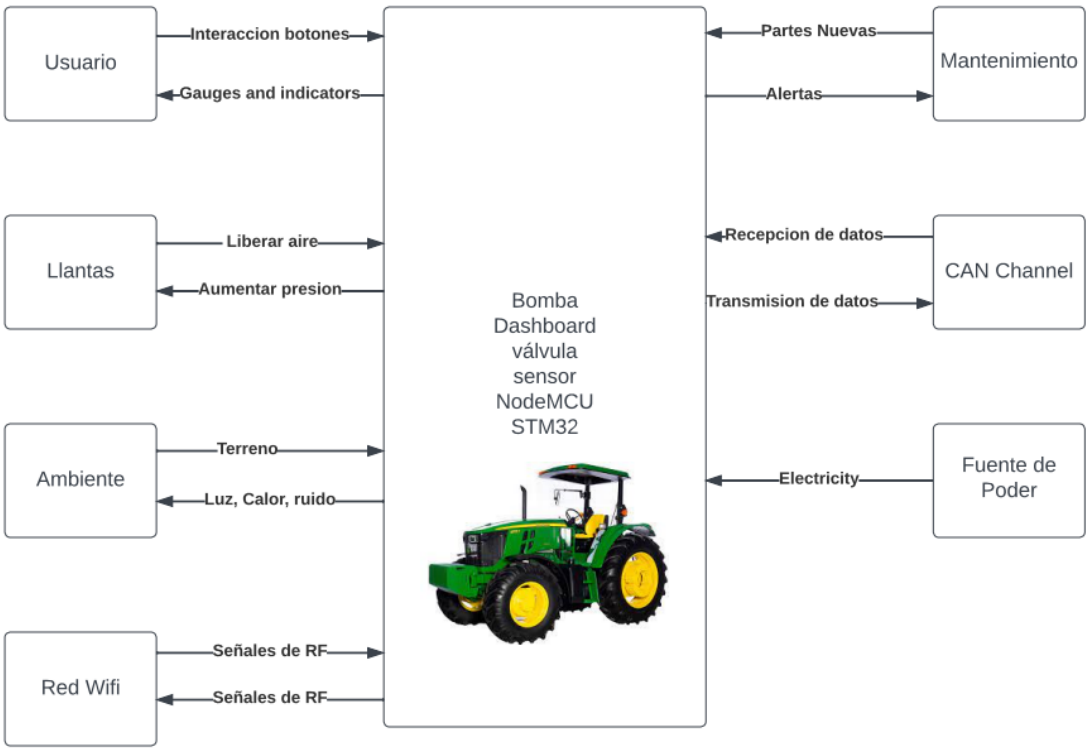


Figure 11. Context Diagram

Requirements Table

Name	**Smart Adaptative Tire Pressure System**
Purpose	**Adjust the tire pressure according**

	to a pre-established setpoint, in response to the user's needs.
Inputs	**Setpoints for type of terrain, Power, Pump Sound**
Outputs	**Tire Pressure, Temperature, light, noise, alert indication**
Functions	**Increase/decrease tire pressure according to type of terrain Generate an alert at dashboard if pump not working properly**
Performance	**Time required for increasing or decreasing the tire pressure to the ideal.**
Manufacturing Cost	**Approx. $3,000 MXN**
Power	**12 v and 3 - 5 A**
Physical size/weight	**80x20x10 cm 500g**

Deployment Diagram

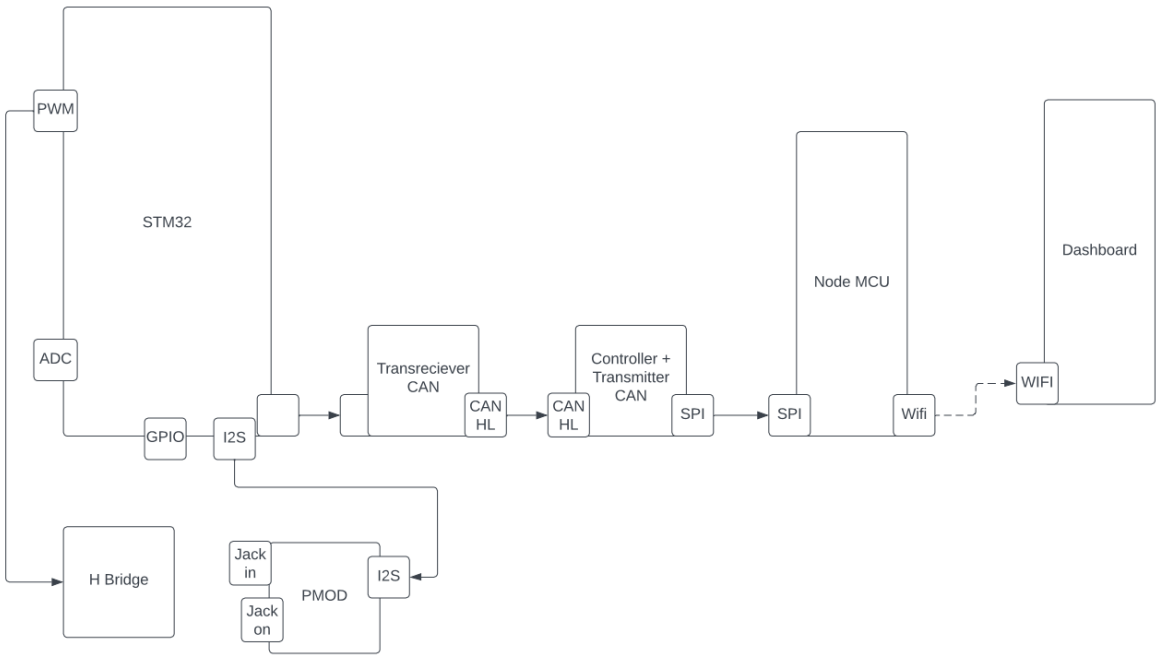


Figure 12. Deployment Diagram

UML Activity Diagrams

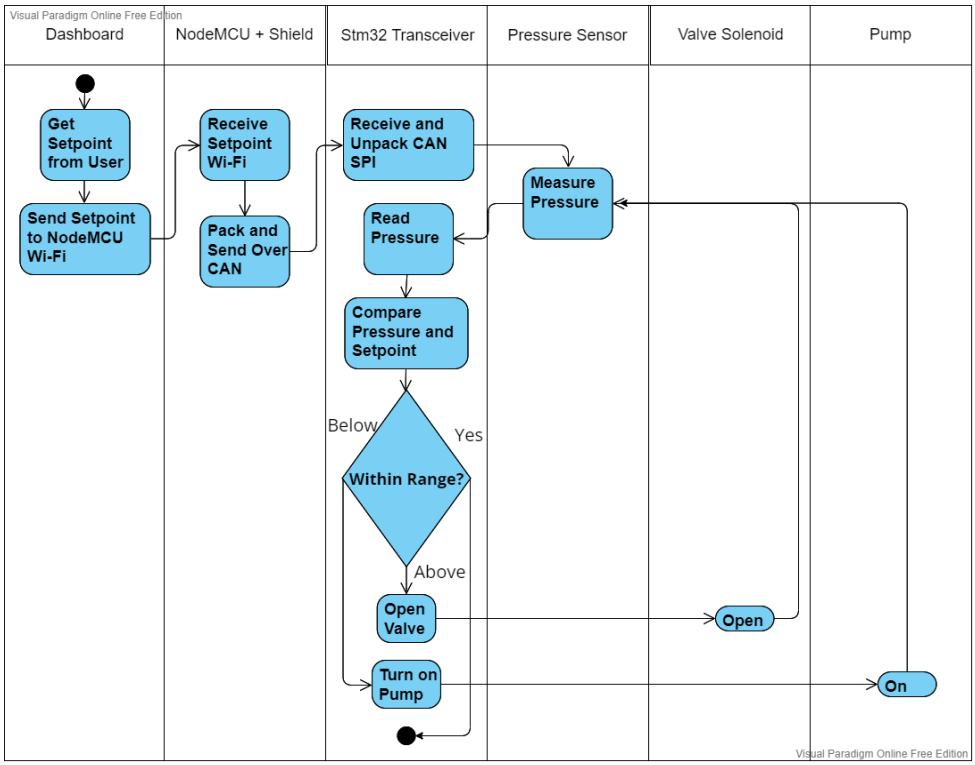


Figure 13. Pressure Adjustment from Terrain Type Diagram

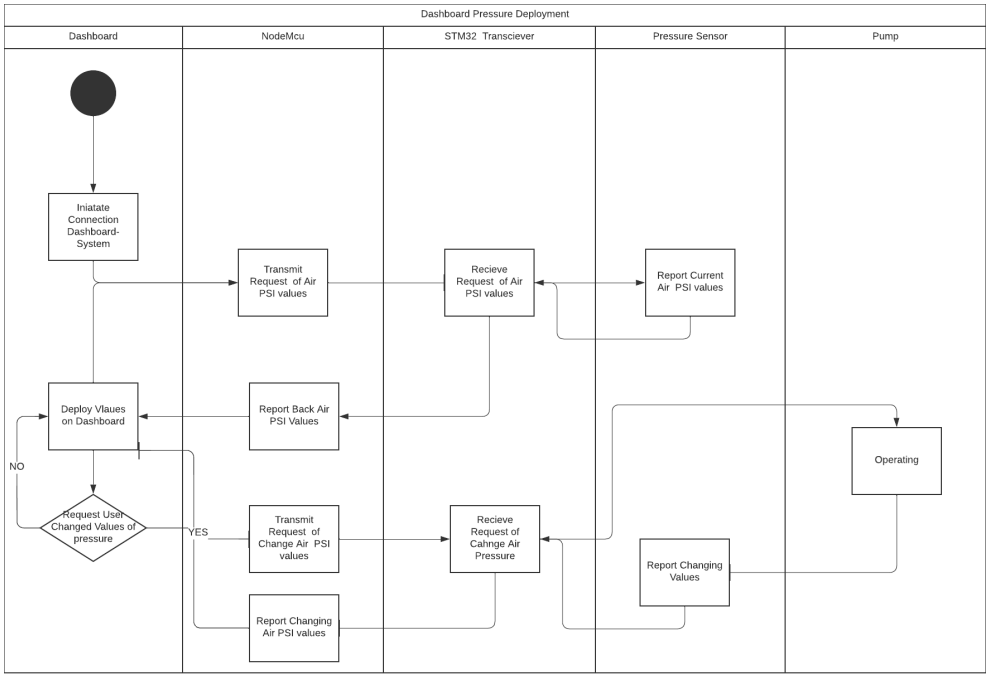


Figure 14. Pressure Display on Dashboard Diagram

Peripheral Devices Selection

Type	Name	Specs
Sensor	P1A Pressure Sensor	Maximum voltage: 5v Maximum Current: 20mA Frequency of Operation: Pressure range from 0 to 10 bar (0-150 psi) Insulation Voltage: 500 VDC
Actuator	Solenoid Valve	Maximum voltage: 12v Maximum Power: 6.5W Maximum Current: 540mA Frequency of Operation: 40 kHz
Actuator	Mikel's NCA-2 Air Compressor 300PSI 12V	Voltage of Operation: 12 v Maximum Air Pressure: 300 psi Current of Operation: 10 A
Periferic	CJMCU-1051 TJA1051	Voltage of Operation: 3v-5v
Periferic	Spi Mcp2515	Voltage of Operation: 3.3v-5v

System Implementation and Tests

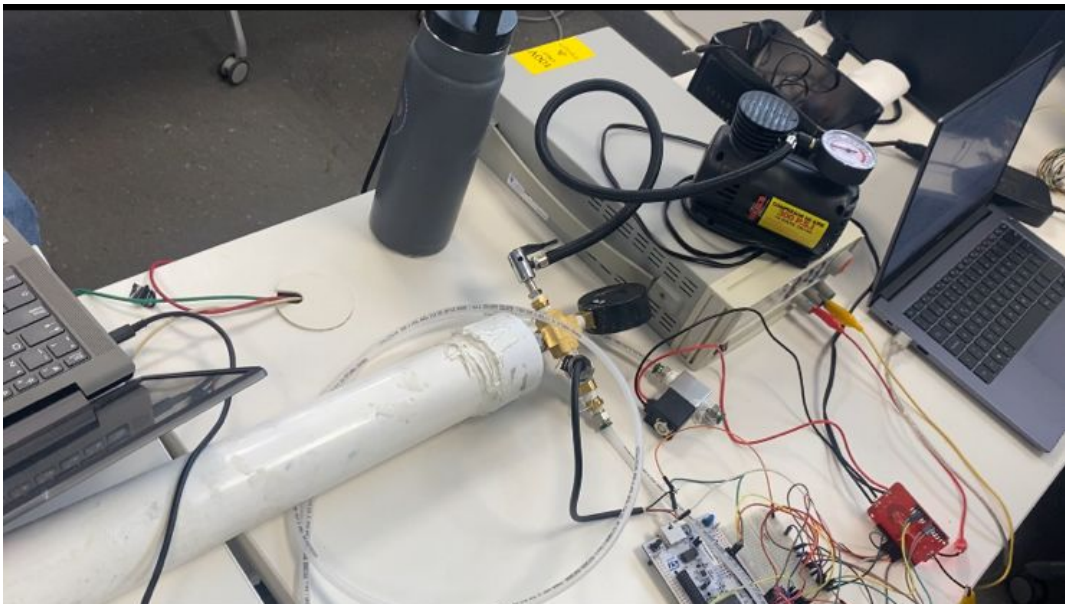


Figure 15. Whole System Photo

Bills of Materials

ID	Name	Designator	Footprint	Quantity	Manufacturer	Manufacturer	Supplier	Supplier Part	Price	Pins	JLCPCB Part C	Contributor	link
1	L293DD	U1	SOIC-20_L12.	1	L293DD	null	LCSC	C9900019186		20	Extended Part	lcsc	
2	MCP2515T-E/	U2	QFN-20_L4.0-	1	MCP2515T-E/	MICROCHIP(微	LCSC	C623189	3.154	21		lcsc	https://atta.szlcsc.comnull
3	TCAN332DR	U3	SOIC-8_L4.9-\	1	TCAN332DR	TI	LCSC	C73651	16.911	8	Extended Part	LCSC	
4	STM32H745Z(	U4	LQFP-144_L2(	1	STM32H745Z(	STMicroelectr	LCSC	C730192	22.067	144	Extended Part	LCSC	

Figure 16. Bills of Materials

Subsystems

Pressure sensor

Implemented in the PF7 pin through ADC3_INP3 with an ADC 16-bit resolution. Code implementation:

```
HAL_ADC_Start(&hadc3);
HAL_ADC_PollForConversion(&hadc3, 100);
raw = HAL_ADC_GetValue(&hadc3);

raw = raw / 100;
raw = raw * 100; //Reduce the amount of noise incoming
psi = (float)raw; //Conversion to float for more precise measure
psi = (psi-10500)/636; //Adjustment through sensor offset and scalar
```

Moving Average

Due to the amount of noise incoming from the readings of the sensor it was necessary to add another layer of filtering to the incoming data.

$$SMA = \frac{A_1 + A_2 + ... + A_n}{n}$$

The moving average smooths the data but the usage of a large number of samples in the calculation produces a significant delay in the readings. Therefore, the implementation in code of simple moving average (SMA) with 10 samples in code:

```
double movingAverage[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

for(int i = 0; i < 9; i++){ //Accumulation of 10 samples
        movingAverage[i] = movingAverage[i+1];
        localAcum += movingAverage[i];
}
localAcum += movingAverage[9];
movingAverage[9] = psi;
psi = localAcum/10;
```

Actuators: Air Compressor and Solenoid Valve

Both actuators are set with a PWM through Timers, the air compressor is implemented through TIM2_CH1 in the Pin PA5 and the solenoid valve is in the TIM3_CH3 at Pin PA6. The code for the actuators mainly relies on two values to change their behavior (PSI read from the sensor and setpoints from the dashboard):

```
if(psi < selectedPSI - 0.1){ // Pressure Values below setpoint
        if(CH1_DC < 65535) CH1_DC = (int)PIDOut; //Compressor On
        if(CH2_DC > 0) CH2_DC = 0;// Valve Off
}else if(psi > selectedPSI+0.7 && psi < selectedPSI + 6.0){ //Pressure Values within a 6 psi margin above the setpoint
        if(CH1_DC > 0)CH1_DC = 0; // Compressor Off
        if(CH2_DC < 65535) CH2_DC = 20000; // Valve On and Off on intervals
        TIM3->CCR1 = CH2_DC;
        HAL_Delay(100);
        if(CH2_DC > 0) CH2_DC = 0;
        TIM3->CCR1 = CH2_DC;
}else if(psi > selectedPSI + 6.0){//Pressure Values 6 psi above from setpoint
        if(CH1_DC > 0)CH1_DC = 0; //Compressor Off
```

```
        if(CH2_DC < 65535) CH2_DC = 20000; //Valve On
}else{ //Case for setpoint mark: both actuators low
        if(CH1_DC > 0) CH1_DC = 0;
        if(CH2_DC > 0) CH2_DC = 0;
}
TIM2->CCR1 = CH1_DC;
TIM3->CCR1 = CH2_DC;
```

Monster Moto Shield WNH2SP30

Component used to interface the actuators to the STM32 board, has an input of 12v and 30 A which presents enough power to operate our devices.

CAN interfacing: CAN Transceiver and CAN Shield

The connectivity of the system is mainly done through CAN protocol, it receives packets without filtering ID's and transmits under the 0x018FEEEA3 id on extended type. The code implementation is:

```
if (HAL_FDCAN_GetRxMessage(&hfdcan1, FDCAN_RX_FIFO0, &RxHeader,
RxData) == HAL_OK){
        HAL_Delay(10);
        printf("\n\rCAN ID: %lx", RxHeader.Identifier);
        HAL_GPIO_TogglePin(LD1_GPIO_Port,LD1_Pin);
        int dataSize = RxHeader.DataLength >>  16;
        printf(" [%x]", dataSize);
        for (int i = 0; i < dataSize; i ++){
        printf(" %x", RxData[i]);
        }
}

if(HAL_FDCAN_AddMessageToTxFifoQ(&hfdcan1, &TxHeader, tireReadings) !=
HAL_OK ){
        Error_Handler();
}
```

PID Controller

The PID controller was calculated according to the identification of the response of the pressure level in the tank. Even though the parameters were taken from the system

behavior, modifications to the coefficients to approximate to unitary values were done to achieve the needed behavior. Code implementation:

```
void PID_Initialize( void ){
   double T = 1;                    //sample rate
   //Complete here with the identified
   //parameters of the plant
   double k = 0.5;
   double tao = 9.2;
   double theta = 1;
   //Calculate the coefficients for
   //the discrete PID controller
   kp = (0.9 * tao) / (k * theta);
   ti = 3.33 * theta;
   q0 = kp + ((kp * T)/(2.0 * ti));
   q1 = ((kp * T)/(2.0 * ti)) - kp;
   q0 = 1; q1 = -1;
}

double PID_Discrete( double yM, double R ){
   e = R - yM;                      //calculate the error
   u = u_1 + q0*e + q1*e_1;         //discrete PID controller
   //Saturate the controller with upper and lower limits
   if(u >= 95.0) u = 95.0;
   if(u <= 0.0) u = 0.0;            //minimum pressure value
   e_1 = e;
   u_1 = u;
   return u;
   }
PIDOut = PID_Discrete((double)psi, (double)selectedPSI);
if(PIDOut > 6) PIDOut = 60000;
else if(PIDOut > 0 && PIDOut < 6)PIDOut = (PIDOut*7500)+20000; //PID output
scaling
else PIDOut = 0;
```

NodeMCU

This is the code to communicate the CAN messages from our STM32 to the Dashboard to reflect values such as the Air in the Tank currently as to even allow changes to the Air pressure if we want it to change through our Dashboard.

```
#include <ArduinoWebsockets.h>
#ifdef ARDUINO_ARCH_ESP32
#include <WiFi.h>
#else
#include <ESP8266WiFi.h>
#endif

#include <mcp_can.h>
#include <SPI.h>

/* Websocket definitions */
const char* ssid = "";//"WIFI_NAME";
const char* password  “"; //Contraseña
const char* serverAddress = "wss://vast-reef-61525.herokuapp.com/:80";


using namespace websockets;
WebsocketsClient client;

/* MCP2515 definitions */
#define CAN_ID_EXTENDED 1
#define WS_BUFFER_SIZE  13
// CAN TX Variables
unsigned long txId = 0;
unsigned char txLen = 0;
unsigned char txBuf[8];
// CAN RX Variables
unsigned long rxId;
unsigned char len;
unsigned char rxBuf[8];
unsigned char rxMsgBuffer[WS_BUFFER_SIZE];
// CAN0 INT and CS
#ifdef ARDUINO_ARCH_ESP32
#define CAN0_INT  21                          // Set INT to pin 2 on ESP32
MCP_CAN CAN0(5);                               // Set CS to pin 5 on ESP32
#else
```

```
#define CAN0_INT  4                           // Set INT to pin 2 on ESP8266
MCP_CAN CAN0(15);                             // Set CS to pin 15 on ESP8266
#endif

void setup() {
  Serial.begin(115200);

  // Connect to WiFi
  WiFi.begin(ssid, password);
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print("*");
  }

  Serial.println("");
  Serial.println("WiFi connection Successful");
  Serial.print("The IP Address of Module is: ");
  Serial.print(WiFi.localIP());// Print the IP address

  delay(2000);

  // Connect to the websocket server
  if (client.connect(serverAddress)) {
    Serial.println("Connected");
  } else {
    Serial.println("CAN Connection failed.");
    while(1) {
      // Hang on failure
    }
  }

  // run callback when messages are received
  client.onMessage([&](WebsocketsMessage message) {
      // Parse ws received data
      const char *wsDataStr = message.c_str();
      // Split data to diff variables
      for (int dataIndex = 3; dataIndex >= 0; dataIndex--)
      {
        txId = (txId << 8) + wsDataStr[dataIndex];
      }
```

```
    txLen = wsDataStr[4];
    memcpy(&txBuf, &wsDataStr[5], sizeof(txBuf));
    // Send messate to CAN module
    byte sndStat = CAN0.sendMsgBuf(txId,CAN_ID_EXTENDED, txLen, txBuf);

    if(sndStat == CAN_OK)
      Serial.println("CAN Message Sent Successfully!");
    else
      Serial.println("Error Sending CAN Message...");
  });

  /* Initialize MCP CAN */
  // Initialize MCP2515 running at 16MHz with a baudrate of 500kb/s and the masks and
filters disabled.
  if(CAN0.begin(MCP_ANY, CAN_500KBPS, MCP_16MHZ) == CAN_OK)
    Serial.println("MCP2515 Initialized Successfully!");
  else
    Serial.println("Error Initializing MCP2515...");

  // Set NORMAL mode
  CAN0.setMode(MCP_NORMAL);

  pinMode(CAN0_INT, INPUT);                            // Configuring pin for /INT input
}

void loop() {
  if(!digitalRead(CAN0_INT))                         // If CAN0_INT pin is low, read receive
buffer
  {
    CAN0.readMsgBuf(&rxId, &len, rxBuf);             // Read data: len = data length, buf =
data byte(s)

    // Send received message to websocket server
    if(client.available()) {
      // Copy data to buffer
      rxId = rxId & 0x1FFFFFFF;
      memcpy(rxMsgBuffer, (unsigned char *)&(rxId), sizeof(unsigned long));
      rxMsgBuffer[4] = len;
      memcpy(&rxMsgBuffer[5], &rxBuf, sizeof(rxBuf));
      // Send buffer - fixed for CAN extended message type
```

```
        client.sendBinary((const char *)rxMsgBuffer, 13);
        Serial.println("WS Message Sent Successfully!");
        Serial.println(rxMsgBuffer[1]);
    }
  }

  // let the websockets client check for incoming messages
  if(client.available()) {
      client.poll();
  }
}
```

Dashboard

This is the code to reflect the values on the dashboard, coded on Javascript to deploy data and with the use of HTML & CSS

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>FARO DASHBOARD</title>
        <link rel="icon" type="image/png"href="icons8-faro-48.png">
        <style>
            #clear, #messages {
                list-style-type: none;
            }
            #newMessage li {
                display: inline;
            }
            #newMessage > li > input {
                text-transform: uppercase;
            }
            .data-input, #newLen {
                width: 20px;
            }
            body {
            background-image: url("Faro.jpg");
            }
        </style>
        <!-- <meta http-equiv="refresh" content="1"> -->
```

```
  </head>
<body>
  <h1>FARO DASHBOARD</h1>
  <ul id="newMessage">
    <li>Id: <input style="width: 70px;" id="newId" type="text" value="18FEEEA3"
maxlength="8"></li>
    <li> Len: <input id="newLen" type="text" value="8" maxlength="1"></li>
    <li> Data: <input class="data-input" type="text" value="3A" maxlength="2"></li>
    <li><input class="data-input" type="text" value="00" maxlength="2"></li>
    <li><input class="data-input" type="text" value="00" maxlength="2"></li>
    <li><input class="data-input" type="text" value="00" maxlength="2"></li>
    <li><input class="data-input" type="text" value="00" maxlength="2"></li>
    <li><input class="data-input" type="text" value="00" maxlength="2"></li>
    <li><input class="data-input" type="text" value="00" maxlength="2"></li>
    <li><input class="data-input" type="text" value="00" maxlength="2"></li>
    <li><button onclick="sendNewMessage()">Send</button></li>
  </ul>

  <ul id="Valuestch">
    <li> PSI cheat-sheet (Change First DATA BOX!): </li>
    <li> 6 psi = 2E </li>
    <li> 12 psi = 34 </li>
    <li> 18 psi= 3A </li>
    <li> 24 psi = 40 </li>
  </ul>
  <ul id="clear">
    <li><button onClick ="clearLog()">Clear Log</button></li>
  </ul>
  <ul style="background: rgba(248,248,255,0.5); font-size: 20px; padding: 10px;
border: 1px solid lightgray; margin: 10px;"id="messages">
    <li id="vals">"Current Value of Sensor: "</li>
  </ul>
  <script>
    var socket = new WebSocket("wss://vast-reef-61525.herokuapp.com/:80");
    //var socket = new WebSocket("ws://localhost:5000");
    socket.binaryType = 'arraybuffer';

    socket.onmessage = function(event) {
      //console.log(event.data);
      let dataView = new DataView(event.data);
```

```
            let dataFrame ="Current Value of Sensor: "
            for(let strDataIndex = 5; strDataIndex < 6; strDataIndex++) {
            dataFrame += " " + ("0" +
dataView.getUint8(strDataIndex)).slice(-3).toUpperCase()/10;
            }
            dataFrame += " psi"
            /*let CANFrame = "Id: ";
            for(let strIdIndex = 3; strIdIndex >= 0; strIdIndex--) {
               CANFrame += ("0" +
dataView.getUint8(strIdIndex).toString(16)).slice(-2).toUpperCase();
            }

         CANFrame += " Len: " + dataView.getUint8(4).toString(16) + " Data: ";
         for(let strDataIndex = 5; strDataIndex < 13; strDataIndex++) {
            CANFrame += " " + ("0" +
dataView.getUint8(strDataIndex).toString(16)).slice(-2).toUpperCase();
         }*/



         /*let messagesList = document.getElementById('messages');
         let newMessageElement = document.getElementsByClassName('li');
         newMessageElement.appendChild(document.createTextNode(dataFrame));
         messagesList.appendChild(newMessageElement);*/
         document.getElementById("vals").innerHTML=  dataFrame;

      };




      function clearLog() {
         document.getElementById('messages').innerHTML = '';
      };
      const fromHexStringToArray = hexString => new
Uint8Array(hexString.match(/.{1,2}/g).map(byte => parseInt(byte, 16)));
      function sendNewMessage() {
         let newDataBuffer = new ArrayBuffer(13);
         let newDataToSend = new Uint8Array(newDataBuffer);
         // Get data
```

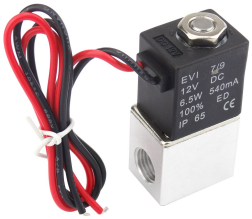
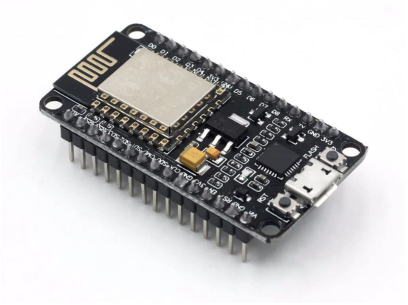
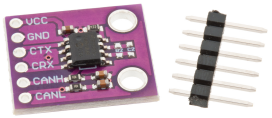
```
        let newId = fromHexStringToArray(document.getElementById('newId').value);
        let newLen = parseInt((document.getElementById('newLen').value), 16);
        let newDataElements = document.getElementsByClassName('data-input');

        // Set id
        let newIdLen = newId.length;
        if(newIdLen == 4) {
            for(dataIndex = 0; dataIndex < newIdLen; dataIndex++) {
                newDataToSend[dataIndex] = newId[newIdLen - dataIndex - 1];
            }
        }
        // Set Len to 8 always (predefined to extended message)
        newDataToSend[4] = 8;
        // Set Data
        for(dataIndex = 0; dataIndex < newDataElements.length; dataIndex++) {
            newDataToSend[dataIndex + 5] =
parseInt(newDataElements[dataIndex].value, 16);
        }

        //Send data
        if(socket.readyState === socket.OPEN) {
            socket.send(newDataBuffer);
        }
        else {
            console.log("WS Client is not connected to server");
        }
    };
  </script>
</body>
</html>
```

Peripheral Devices

Solenoid Valve	NodeMCU

	
Pressure Sensor	Air Compressor
	
Monster Moto Shield VNH2SP30	CAN Transceiver and Shield
	 

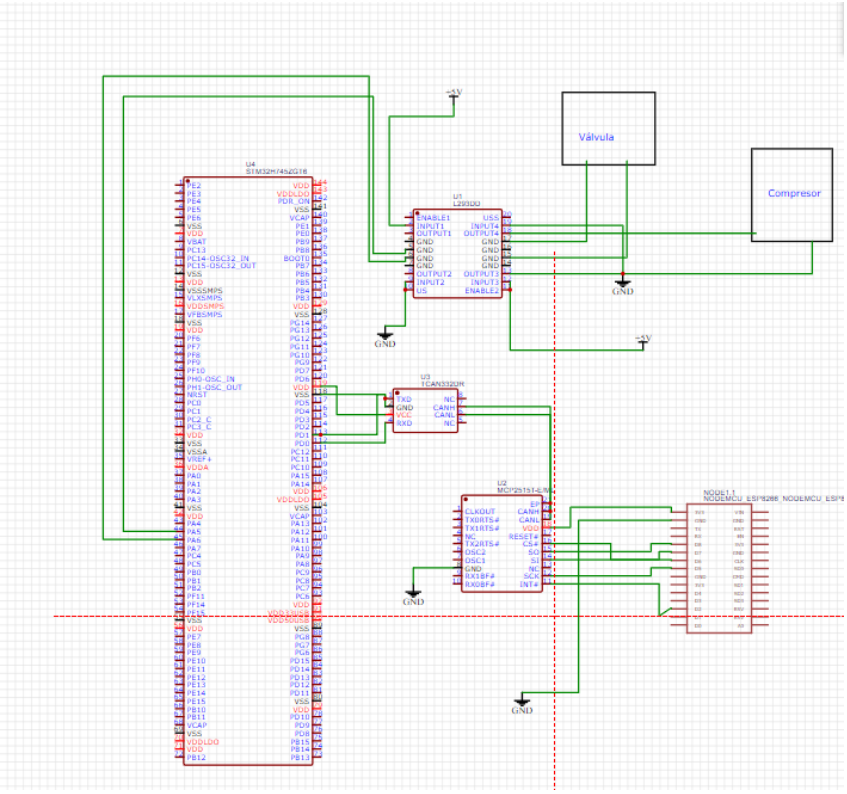


Figure 17. Schematic Diagram

Link to STM32CubeIDE project:
https://drive.google.com/drive/folders/13bYtNqAslGtItx8tYks4rUnxx1qcuNYe?usp=share_link

Testing

Integration Test

	Pressure Analysis/23-Nov-2022			
Team	FARO			
Responsible	Arturo José Murra López José Alfredo García Alvarado			
Description	Test integration of the ADC readings of the sensor and display the correct value of pressure in the serial port.			
Function	Monitor temperature (use case – functionality)			
Components	Uart driver, UART, measure control (main).			
Hardware	STM32 board, Analog Pressure Sensor, computer with serial interface, Air compressor			
Procedure		Expected behavior	Actual behavior	Result
	Initial conditions	Empty values of pressure displayed in serial	Oscillation between 0 and 1 PSI.	
	Step1: Rise pressure level	Pressure must increase in the serial display	Values increasing as the air compressor fills the tank	OK

Comments	The procedure followed to adjust the raw value from the sensor is escalating the value to adjust the offset of the readings and then adjust it in proportion to the maximum value of the sensor.

Link to demonstration video:https://drive.google.com/file/d/1elmzBsQrfOKlkNMU8R92jgpW2_KqZPIm/view?usp=share_link

Unit Test

	Serial Interfacing/22-Nov-2022			
Team	FARO			
Responsible	Oscar Cárdenas Gómez Ricardo Gonzalo Cruz Castillo			
Description	Test that the serial port is correctly sending a message to the Putty			
Function	Report the behavior of specific parts of the system through the serial port			
Components	Uart driver			
Hardware	STM32 board, computer with serial interface, extension cable			
Procedure		Expected behavior	Actual behavior	Result
	Initial conditions	Serial monitor is empty	Serial monitor is empty	OK
	Step1: Begin program	Message “Prueba Envío de Datos” displayed on screen	The message match the expected value	OK
Comments				

Link to demonstration video:
https://drive.google.com/file/d/1GmgVbiwcwglDjz5M_NmH2Xzxpq1L7vHC/view?usp=share_link

System Test

	Pressure Auto-adjustment/29-Nov-2020			
Team	FARO			
Responsible	Arturo José Murra López José Alfredo García Alvarado Oscar Cárdenas Gómez Ricardo Gonzalo Cruz Castillo			
Description	Modify the value of a setpoint in the dashboard to have a pressure adjustment in the system.			
Function	Change pressure according to setpoint			
Components	Arduino IoT, NodeMCU, CAN interface, ADC, PWM			
Hardware	STM32 board, NodeMCU, CAN Shield, CAN Transceiver, H-Bridge Monster, Air compressor, Solenoid Valve, Pressure Sensor			
Procedure		Expected behavior	Actual behavior	Result
	Initial conditions	System loads default setpoint mark (18 psi) and adjust itself to its value	The pressure reading shows a value of 18.1	OK
	Step1: Change setpoint to 6	System loads setpoint mark from CAN (6 psi) and adjust itself to its value	The pressure reading shows a value of 6.02	OK
	Step2: Change setpoint to 24	System loads setpoint mark from CAN (24 psi) and adjust itself to its value	The pressure reading shows a value of 24.01	OK

	Step3: Change setpoint to 12	System loads setpoint mark from CAN (12 psi) and adjust itself to its value	The pressure reading shows a value of 11.989	OK
Comments	When decreasing pressure, the system adjustment becomes less exact.			

Link to demonstration video:
https://drive.google.com/file/d/1CsIM7DWIvsmYQ-yMPSHUUXg64eWkWLvb/view?usp=share_link

Conclusión

The goal of this project was to give the training partner, John Deere, a solution according to the requirements they specified. John Deere asked for a smart pressure system that would adapt the psi within their vehicles' tires depending on the terrain it is being driven on. For this we made an air container, simulating the fire, with a PVC schedule 40 pipe. We integrated an RTOS composed of two actuators and one sensor, which were an air compressor and a solenoid valve. We also created a dashboard for updating the setpoint in which the tires' pressure should be regulated at and delivered that information to the STM32H7 development board with a NodeMCU which allowed us to use WiFi to CAN communication. Depending on the setpoint sent via WiFi, the pressure on the air container is regulated by enabling the air compressor or the solenoid valve. The purpose of developing this project was to make a system that would take into account the environmental variables in its surroundings in order to function at optimal capacity.

The main biggest achievement of this assignment was resilience. There were a lot of setbacks throughout the whole development process. The initial air container had leakages and it was a safety hazard, thus it was decided to scratch it and make one that was better suited and had better materials. Also, the initial standard platform in which we were supposed to make our dashboard did not want to connect to the NodeMCU, therefore we created our own dashboard with HTML which we were able to establish communication with. The sensor was also really inconsistent in its readings, for this we implemented a moving average so that it would behave less erratically and we also implemented a PID controller. There was a lot of problems with this last software implementation but it was ultimately implemented correctly.