UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

ESCUELA TÉCNICA SUEPERIOR DE INGENIERÍA DEL DISEÑO

UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEVELOPMENT OF THE CONTROL ELECTRONICS FOR THE NAVIGATION OF AN UNMANNED SUBMARINE WITH ARDUINO

TRABAJO DE FIN DE GRADO:

Grado en Ingeniería Electrónica Industrial y Automática

Autor: Fco. Javier Pérez Villaplana

Director: Jose Vicente Busquets Mataix

Codirector: Javier Busquets Mataix

VALENCIA, ABRIL 2019

RESUM

L'objectiu principal d'aquest projecte és el disseny dels sistemes electrònics involucrats en el control d'un Vehicle Autònom Subaquàtic (VAS) fent ús del microcontrolador Arduino MEGA 2560 com a plataforma de desenvolupament, assegurant la fiabilitat i el baix cost dels sistemes mencionats. Com es habitual en la forma de treballar amb Arduino, el projecte estarà organitzat en diferents mòduls cadascú amb una funció específica dins del conjunt de sistemes de control.

Dintre del projecte, es trobarà la caracterització de sensors i actuadors, el estudi del diferents mòduls Arduino, la integració d'aquests mòduls, el disseny de l'estructura de control del VAS, així com la implementació y muntatge del mòduls al xassís del vehicle.

Els principals sistemes a controlar i integrar son: control de rumb, sistemes de visualització i comunicació, emmagatzemament de informació i sistemes de representació de resultats. Com es tracta d'un projecte multidisciplinari, diversos estudiantes de l'ETSID i l'ETSINF estaran al càrrec del disseny d'altres sistemes fonamentals del VAS que s'uniran als sistemes inclosos en aquest projecte.

El codi de control desenvolupat permetrà un correcte funcionament del VAS mitjançant la toma de autònoma de decisions, en qualsevol situació, gràcies a les lectures provinents del diferents sensors. La monitorització d'aquests sistemes es de gran importància ja que permet assegurar que tot està sota control i que el vehicle està navegant dins del seu règim normal de funcionament.

Paraules Clau: VAS, Arduino, Sistemes de Control, Control de Rumb

RESUMEN

El objetivo principal de este proyecto es el diseño de los sistemas electrónicos involucrados en el control de un Vehículo Autónomo Subacuático (VAS) mediante el uso del microcontrolador Arduino MEGA 2560 como plataforma de desarrollo, primando el bajo coste y la fiabilidad de dichos sistemas. Como es habitual a la hora de trabajar con Arduino, el proyecto se organizará de manera modular, donde cada módulo realiza una función diferenciada dentro del conjunto de sistemas de control.

Dentro de este proyecto, se encontrará la caracterización de sensores y actuadores, el estudio de los diferentes módulos Arduino, la integración de dichos módulos, el diseño de la estructura de control del VAS, así como la implementación y montaje de dichos sistemas en el chasis del vehículo.

Los sistemas principales a controlar e integrar son: control de rumbo, sistemas de visualización y comunicación, almacenamiento de datos y sistemas de representación de resultados. Dado que se trata de un proyecto multidisciplinar, varios estudiantes de la ETSID y la ETSINF se encargarán de diseñar otros sistemas fundamentales del VAS que se unirán a los sistemas incluidos en este proyecto.

El código de control desarrollado permitirá el correcto funcionamiento del VAS mediante la toma autónoma de decisiones, en cualquier situación, gracias a la interpretación de las lecturas provenientes de los sensores. La monitorización de todos estos sistemas es de gran importancia dado que permite asegurar que todo está bajo control y que el vehículo está navegando dentro de su régimen normal de funcionamiento.

Palabras Clave: VAS, Arduino, Sistemas de Control, Control de Rumbo

ABSTRACT

The main goal of this project is to provide a low-cost and reliable design for the electronics systems involved in the control of an Autonomous Underwater Vehicle (AUV) using Arduino Mega 2560 as a development platform. The project will be organized in a modular fashion as it is the usual way of working with Arduino, where each module performs a different task.

The project's scope encompasses the characterization of the actuators and sensors, the study of the different modules, the integrations of all the modules, the design of the control structure and the implementation of the systems, including the mounting on the AUV chassis.

The main systems to be controlled and integrated are: attitude control systems, visualization systems, communications, data storage and representation systems. As this is a multidisciplinary project, several students from the ETSID and ETSINF schools will be in charge of designing other vital systems of the AUV that will also be merged with the ones explained in this project.

The control code written will allow for the proper operation of the AUV vehicle by combining every trace of data coming from the sensors into useful information in order to actuate appropriately in every situation. Monitoring every subsystem is vital to ensure everything is under control and working in a normal regime.

Keywords: AUV, Arduino, Control Systems, Attitude Control

Agradecimientos

Quiero agradecer, en primer lugar, a mis padres por su apoyo incondicional a lo largo de lo todos los años de carrera y por permitirme la gran oportunidad de cursar mis estudios fuera de casa sin privarme de ninguno de los aspectos que han hecho de vida estudiantil una etapa inolvidable.

En segundo lugar, quiero dar las gracias a mi pareja, Marlene, por ayudarme y animarme en todo momento, impidiendo que me rindiera en la realización de este proyecto hasta el último minuto. Sin ella, este trabajo no sería más que una ilusión esperando a ser redactada.

Finalmente, quiero agradecer a todos mis amigos y compañeros de carrera por hacer de esta etapa algo especial.

DOCUMENTS

1. Report
2. Requirements
3. Budget
4. Diagrams & Schematics
5. Annexes

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

ESCUELA TÉCNICA SUEPERIOR DE INGENIERÍA DEL DISEÑO

UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEVELOPMENT OF THE CONTROL ELECTRONICS FOR THE NAVIGATION OF AN UNMANNED SUBMARINE WITH ARDUINO

1. REPORT

TRABAJO DE FIN DE GRADO:

Grado en Ingeniería Electrónica Industrial y Automática

Autor: Fco. Javier Pérez Villaplana

Director: Jose Vicente Busquets Mataix

Codirector: Javier Busquets Mataix

VALENCIA, ABRIL 2019

Index

Figure Index

1.- Introduction

1.1.- Contextualization

The idea of Autonomous Underwater Vehicles (AUV) has been around since 1989, when H. Stommel first proposed the design, and later prototype, of an unmanned underwater vehicle based on the variable buoyancy principle. This way, AUV convert part of their buoyancy force into surge velocity, thus taking advantage of the Archimedes principle. To do so, they must follow a sawtooth navigation pattern and should be provided with systems that are able to produce an increase/decrease of the vehicles' volume (water-air ballast, oil-air buoyancy control) to induce a change in its net buoyancy.

As AUV are only propelled by this principle and not via motors, it could be said that they glide through the water as much as their aviation counterpart glide through the air. For this reason, some of these AUV are also called Gliders. Through the years, many different versions of AUV have been proposed which included electric propellers [1], lube-oil filled bladders [2] or jet pump propulsion [3]. Besides, different strategies, such as taking advantage of the ocean's temperature [4] or underwater currents [5] have been studied in order to reduce energy consumption and increase the range of the Gliders. Some these designs have been a commercial success and have been used in several oceanographic explorations and missions, including a successful trip across the Atlantic [6].

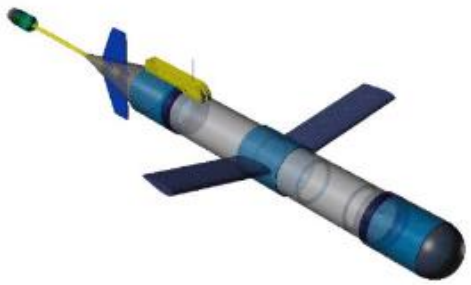


Fig. 1: *ALBA 14 HGL Glider 3D modelling*

Given the success of these vehicles, from the UPV, a new design, intended to reduce the cost of fabrication of the Glider while increasing its range and navigation capabilities, was proposed. These series of low-cost AUV was called ALBA Gliders and it has been in continuous development for several years by UPV students and professors. The last version of the ALBA Glider, which this project is focused on improving, is the ALBA-14 HGL Glider [7]. This vehicle (**Fig. 1**[1]) is designed in a modular fashion, using Arduino as a developing platform, and it includes a hybrid buoyancy system which combines compressed air and oil to fill the Glider's bladders.

[1] Busquets J., Busquets D.., Busquets J.V., "Combined Gas-Fluid Buoyancy System for Improved Attitude and Maneuverability Control for Application in Underwater Gliders", *IFAC-PapersOnLine*, 48-2 (2015) 281–287

1.2.- Motivation

Given the unstoppable increase of chemical spills and other contaminants that end up in our seas and oceans, underwater exploration and sampling has become increasingly more important. It helps predict biological disasters or disturbances in the natural balance of the sea fauna and flora such as plagues or water contamination. The main motivation for this project is to provide sea scientist and marine biologist with a low-cost, highly efficient underwater glider that reduces the cost of these explorations.

Personally, I find this project very interesting as its main goal is to combine different control, visualization and data recording systems into one control structure. Fulfilling this task allows me to characterize sensors and actuators, design control strategies and interface different modules and systems together. Besides, it helps me to further develop my coding abilities as the control is implemented using an Arduino Mega board. For these reasons, I think that this project is the perfect combinations of every skill learned in the degree I studied: electronics, control engineering and IT. Thus, I want to prove my knowledge and skills in my field of study through this project.

1.3.- Objectives

The objective of the project is to design a solution for the attitude systems of the AUV in charge of keeping the vehicles pitch and roll angles within the user's specifications as well as designing the visualization, communication and data storage systems that help the user know the status of the vehicle in real time. The integration and mounting of all the different subsystems will also be assessed together with explanations on the Arduino control code written.

2.- Planning

The approach chosen for the completion of this project, given the fact that it is Arduino based, is a modular approach. Thus, each different sensor and module will be coded and tested independently and then merged into the main design.

First, an initial study of the previous Glider was performed to detect the main flaws of the mass trimming systems which is one of the main points of this project. Then, a hardware selection for the new system is carried out including motors, sensor, controller and mounting boards.

With the new stepper motors selected, it was found that the controller needed to operate them was not manufactured as an Arduino shield and thus a custom PCB was designed to accommodate the new controllers.

Once the hardware and control strategy for the mass trimming system were defined, Bluetooth communication (BT) was next. In order to keep the vehicle monitored and under control, new commands for the control system were coded and then included in a custom APP designed for the exchange of data between the Glider and the mobile device.

Then, a screen display module was added to the project which allowed for the display of Real Time Data (RTD) when setting up the Glider on the water surface. The I2C communication between different Arduino board was also assessed at this point.

Following this, another Arduino board, in charge of collecting sensor data coming from a CTD module, was added to the I2C bus. The proper communication between boards was tested separately and then together with the rest of the I2C devices (accelerometer, RTC, etc…).

Next, an SD Card module using SPI communication is tested and added to the final design together with an RTC module connected via I2C. The combination of these two, together with the sensor data coming from the different sensors in the Glider, allow for the creation of a real time data log which was also tested and validated.

Finally, every module was mounted on the back of the mass trimming system structure and proper function of the different modules was tested when working together.

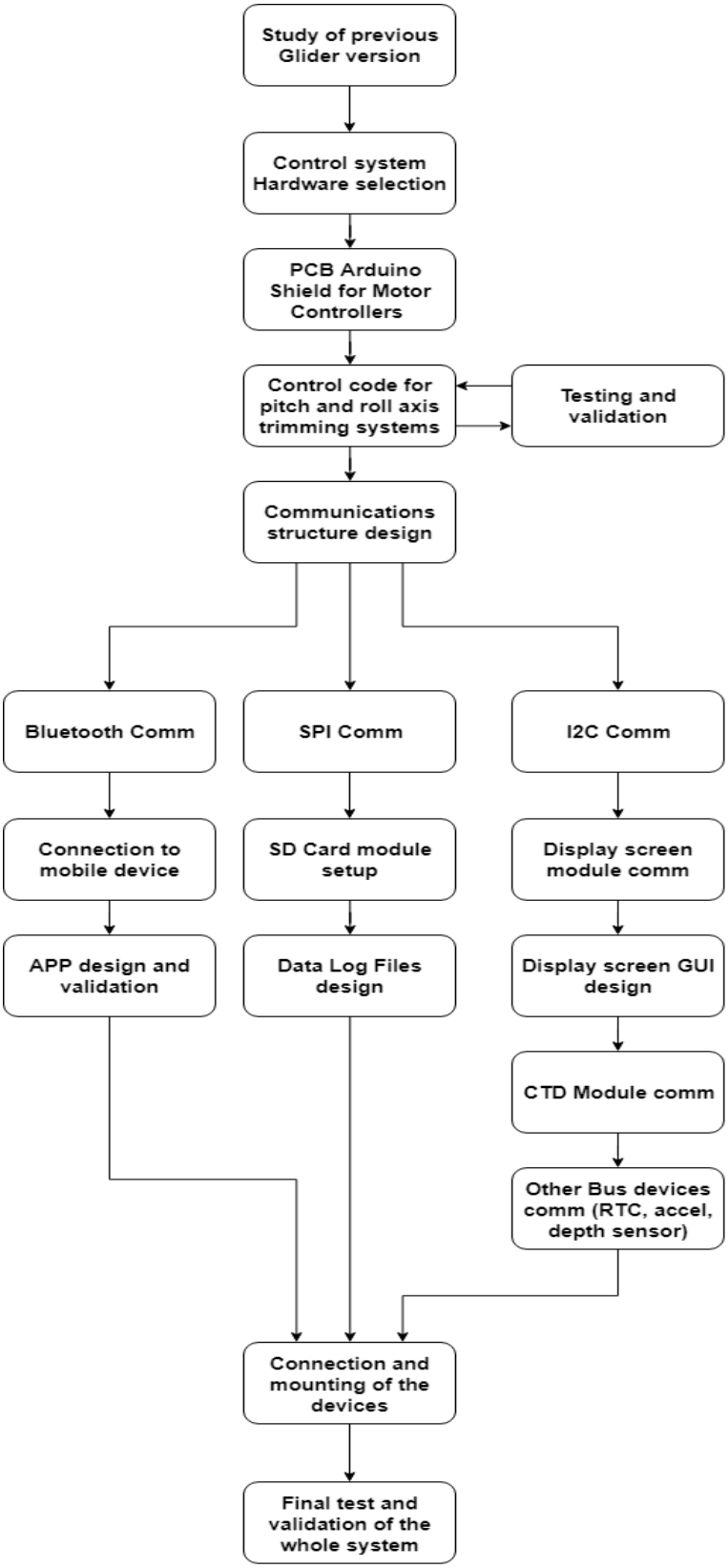


Fig. 1.1: *Project Planning Diagram*

3.- Attitude control subsystems

3.1.- Introduction

The aim of this part is to explain the design, implementation and testing of the attitude control subsystems incorporated in the Alba glider. Working together with the oil-air buoyancy control and the water-air ballasts, these attitude control systems ensure that the pitch and roll angles of the Glider follow the ones selected by the user within a certain margin.

To do so, these systems are in charge of moving two inertial masses big enough to change the centre of mass (CM) of the vehicle and create a positive or negative torque depending on the direction of movement of these masses. Thus, this part will be divided in two subparts related to each of the vehicle's axes:

a) **Pitch axis**: The control of the pitch angle is performed by the back or forth movements of the mass in a lineal motion along the longitudinal axis of the vehicle.
b) **Roll axis:** The control of the roll angle is performed in a rotational fashion where the mass revolves around the longitudinal axis of the vehicle.

The use of stepper motors is common to both systems. This allows for a precise control of the position of the masses in open loop. The motors are connected to an endless screw, which is then mashed with a gear suited for each application, whether it is lineal or rotational motion. Further explanations about the design and implementation of these systems will be provided within this section.

3.2.- Previous work

In previous versions of the Alba Glider, the attitude control system was implemented using DC motors which tend to be bulky, heavy and imprecise. These characteristics are not desirable in an underwater vehicle as the weight and instability are increased with no apparent upside on torque or power from the motor. Furthermore, after testing, it was found that these DC motors were short on torque when the mass was pulled against gravity and compensation for this phenomenon had to be taken into account.

To avoid this problem and improve the overall performance of the vehicle, stepper motors will be used. The program created to run the vehicle with the DC motors (Glider 1) will be used and modified so as to control the new motors.

This first part focuses on the modifications performed on the "Glider 1" code and the measurements, test and calculations done in order to control the stepper motors properly while improving the performance of the trimming systems and the code.

3.3.- Hardware Selection

The very first part in order to design a control system is to know all the technical details about the systems to be controlled. For this attitude control subsystems, two main hardware components will be used: the stepper motor and the motor controller. These will be explained in this section.

3.3.1.- Stepper Motor

The main concern when choosing the right motor is to fulfil the torque and power demands required to move the masses along the Glider. After comparing technical data of several stepper motors, the NEMA 17HS15 was chosen (**Fig. 2**). This motor includes a planetary gearbox which helps deliver a higher torque at a slower speed as seen in the datasheet of the motor (**Fig. 4**). This feature allows the motor to move the mass for and against gravity without losing any step and thus keeping a precise control of the mass position.

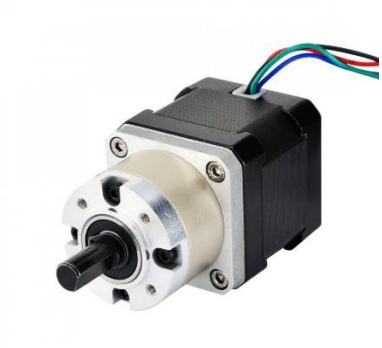


Fig. 2: *NEMA 17 Stepper Motor*

CCW

STEP	A	B	C	D
1	+	+		
2		+	+	
3			+	+
4	+			+

CW

Fig. 3: *Stepping Sequence*

Voltaje nominal	2.8 V
Corriente / fase clasificada	1.68A
Resistencia de fase	1.65ohms
Inductancia	4.1 mH
Max. Par estático	4.4kg-cm(62 Oz-in)
Conductores	4
Proporción	1:71
Inercia rotacional	68 g-cm2

Fig. 4: *Stepper Motor Datasheet*

After testing the motor thoroughly with the right controller, it was proved that the motor can fulfil the task without any losses. Furthermore, using a heavier load does not affect the performance. This is important in order to ensure proper function in a real-world scenario.

3.3.2.- Motor Controller

Due to the size and characteristics of these motors, it is impossible to drive it using directly the Arduino Mega controller as it needs an external power source and a control circuit that will provide power to each of the phases in the right order for the motor to step (**Fig. 3**). Performing this operation via coding would consume a lot of processing power from the Arduino Mega, slowing the process down.

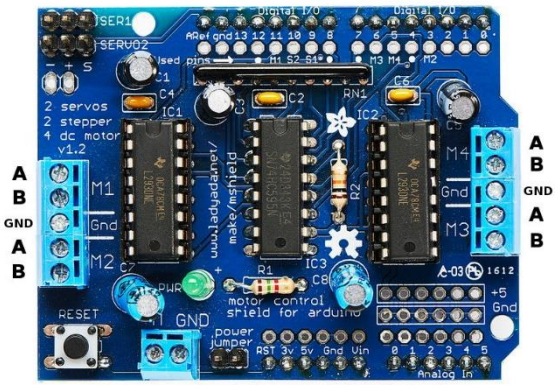


Fig. 5: Adafruit Motor Controller

However, choosing the right controller is a daunting task. The first option tested was to use the Adafruit Motor Shield [8] based on the **L293** IC (**Fig. 5**). This controller is intended for the control of 4 DC motors simultaneously, or 2 stepper motors with a 4-phase configuration such as the NEMA 17. In order to test this controller, the motor was hooked up according to the data sheet and tested with a simple Arduino program using the "AFMotor.h" library main instructions (**Fig. 6**).

```
// Connect a stepper motor with 200 steps per revolution (1.8 degree)
// to motor port #2 (M3 and M4)
AF_Stepper motor(200, 2);

motor.setSpeed(120);  // 120 rpm
motor.step(92, FORWARD, SINGLE); //the motor moves the tray exactly 1mm back or forth
```

Fig. 6: "AFMotor.h" library motor control instructions

As it can be seen, using the library makes the code simple. The only input values used to control the motor are the motor speed, the direction, the stepping mode and the number of steps to be performed. The controller will provide power to the motor and perform the stepping on its own. Despite all of this, several problems were found during the test of this controller:

- Overheating of the IC for the demanded current (**1 A max output current**).
- Step control not precise enough.
- Large amount of vibrations when the motor was operating.

Two options were tested as a solution for these problems:

- Soldering an additional IC on the top of the first one to increase the current output (Piggyback).
- Search for a controller designed specifically for stepper motors of these characteristics.

The first option was discarded as the overheating problem persisted after soldering the additional IC. Besides, the solution looked clumsy, fragile and prone to fail at any moment. Thus, the second solution was adopted.

To do so, different stepper motor controllers were compared in order to choose the most appropriate one. The main concern was to avoid the overheating of the IC while providing enough current to the motor. The controller chosen and used for both motors is a breakout board based on the **DRV8825** IC from Polulu [9] (**Fig. 7**).

This breakout board is designed specifically for stepper motors control and provides a maximum current output higher than the one demanded by the motor (1.68 A) thus avoiding the overheating. The main characteristics of this controller are the following:

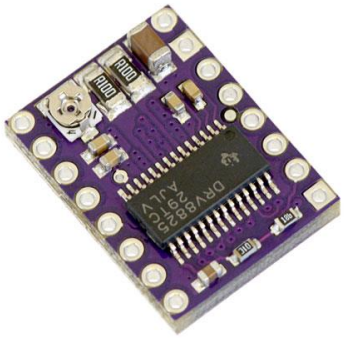


Fig. 7: DRV8825 Polulu's Breakout Board

- Simple step and direction control interface (2 pins).
- Six different step resolutions: Full step up to 1/32-step.
- Adjustable current control to set the maximum current output avoiding any damage to the motor when using voltages above the motor's rated voltage.
- 45 V maximum supply voltage.
- Built-in regulator.
- Over-temperature thermal shutdown, over-current shutdown, and under-voltage lockout.

Due to the fact that this little breakout board was not integrated into an Arduino shield, compared to the Adafruit L293 controller, and given the necessity to use two controllers, one per motor, a brand new PCB was designed in order to house both controllers and act like a shield for the Arduino Mega. The next section will provide in-depth detail of the design and fabrication processes of the PCB.

DRV8825 Arduino Shield PCB Design

Keeping in mind the concept of modularity and tidiness, it was decided to design a PCB to integrate both controllers as an Arduino shield thus reducing the number of soldering points and cables needed. The design was performed using a free development PCB web site called "CircuitMaker" [10] developed by Altium [11]

The first part in the design is to know the basic connections and control signals needed to control the motors stepping using the DRV8825 controller (**Fig. 8**).

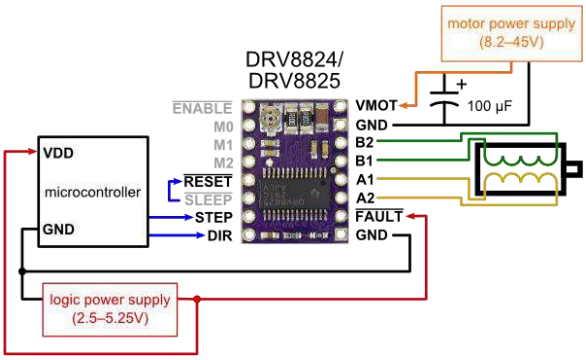


Fig. 8: DRV8825 Basic Connections

As it can be seen, only two pins are necessary to control the motor's stepping (STEP & DIR), M0 to M2 are used to select the stepping mode, an enable pin should also be used activate the IC and four pins are used for each of the phases of the bipolar motor.

Based on these connections, the electric layout of the PCB is designed (**Fig. 9**) taking into account the following features of the controller:

- Stepping mode selection through 3 DIP Switches. This way, the connections are hardwired and no pins of the Arduino Mega are wasted on mode selection.
- Decoupling capacitors close to the breakout boards and on the motor voltage input.
- Hardwired RESET, SLEEP pins as they should always be connected to Vcc.
- ENABLE pin wired to an Arduino pin to control the activation of the IC.

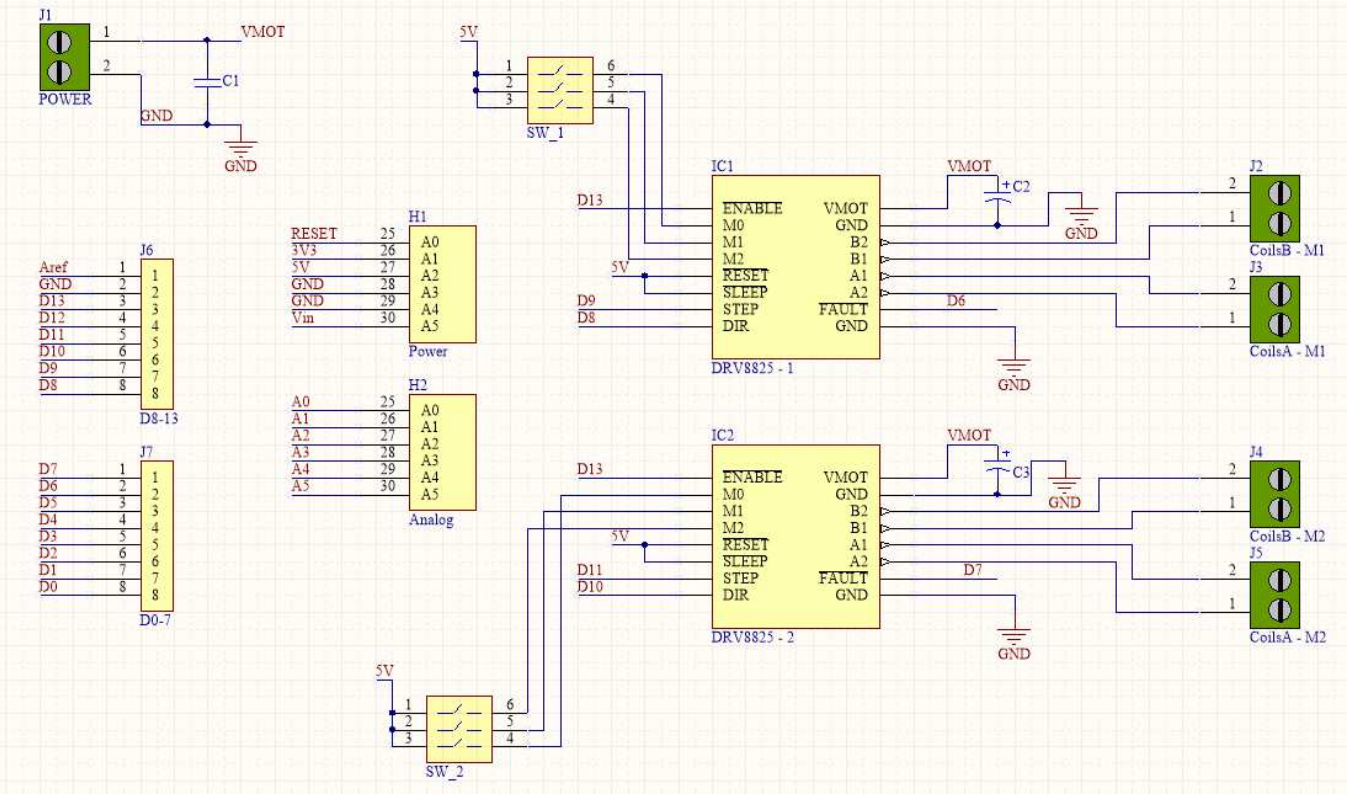


Fig. 9: PCB Electric Schematics

As seen in the schematics, the Arduino Mega pins will be connected to the jumpers J6, J7, H1 and H2 named as 'D8-13', 'D0-7', 'Power' and 'Analog' respectively. The connections of both controllers to the Arduino pins are the following:

Controller 1

- ENABLE ->> D13
- M0-M2 ->> Hardwired to DIP Switch
- RESET & SLEEP ->> Hardwired to Arduino 5V Pin
- STEP ->> D9
- DIR ->> D8
- FAULT ->> D6

Controller 2

- ENABLE ->> D13
- M0-M2 ->> Hardwired to DIP Switch
- RESET & SLEEP ->> Hardwired to Arduino 5V Pin
- STEP ->> D11
- DIR ->> D10
- FAULT ->> D7

The rest of the connections are the external power source and the phases of the stepper motors. As these pins carry the highest intensity and voltage in the circuit, the traces of the PCB will be thicker and the jumpers, J1 to J5, have a small screw in order to secure the connection thus avoiding damage to the motors or the power supply unit. The decoupling capacitors are added between the main power lines so as to avoid bouncing and voltage or current peaks.

Following these electric schematics, the footprint of each of the components is included (**Fig. 10 – 15**) as they would appear on the PCB design (**Fig. 17**). To design the PCB with the exact dimensions of an Arduino Mega shield, the official measurements and shape for an Arduino shield was used (**Fig. 16**). After cutting out the board to the right dimensions and shape, the components are laid out.

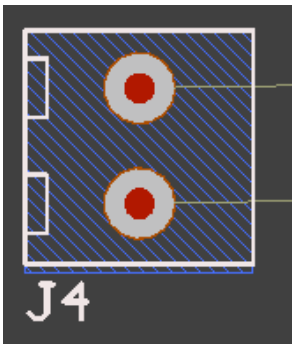


Fig. 10: J1-J5 Jumper PCB

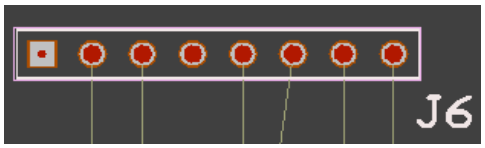


Fig. 11: J6-J7 Jumper PCB Footprint

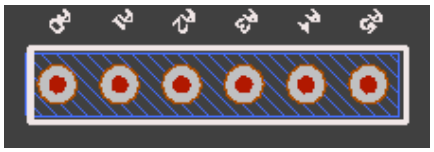


Fig. 12: H1-H2 Jumper PCB Footprint

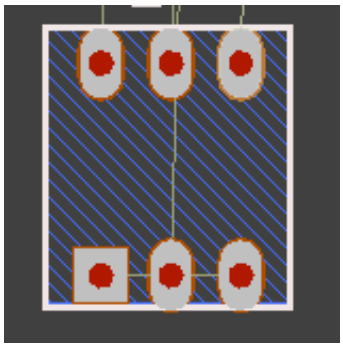


Fig. 13: 3 DIP Switch PCB Footprint

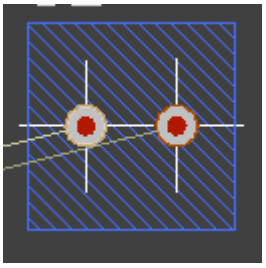


Fig. 14: Decoupling Capacitors PCB Footprint

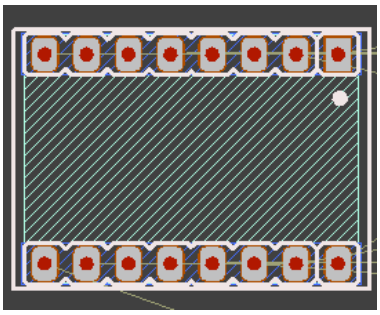


Fig. 15: DRV8825 PCB Footprint

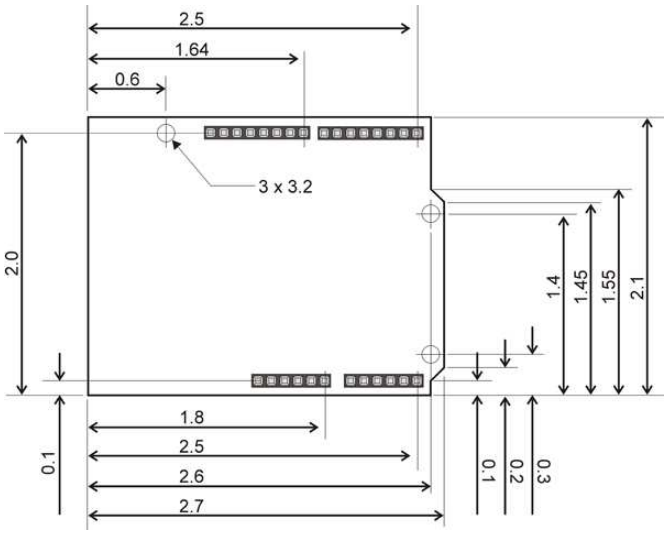


Fig. 16: Arduino shield measurements (inches)

The final disposition of the components (**Fig. 17**) is achieved after several tries with the "autoroute" tool in order to ensure that all of the traces were at least 20 mil (inches) thick and every connection was made without a significant amount traces overlapping.

Once the connections are validated and the routing of the traces is completed successfully, all of the Gerber Files needed for the fabrication (**Fig. 18**) of the PCB are generated using the tool designed for that purpose on 'CircuitMaker'.

Due to the low cost of fabrication, the quality reports from other users and shipping fees applied, the PCB was fabricated using the Chinese factory called 'Seeed Studio' [12] with the following fabrication options:

- Surface Finish: HASL
- Minimum Solder Mask Dam: 0.4mm
- Copper Weight: 1 oz.
- Castellated Holes: No
- Minimum Drill Hole Size: 0.3mm
- Colour: Blue
- Base Material: FR-4 TG130
- 2 layers
- Quantity: 10
- Buried Vias: No
- Thickness: 1.6 mm
- Spacing: 6/6 mil

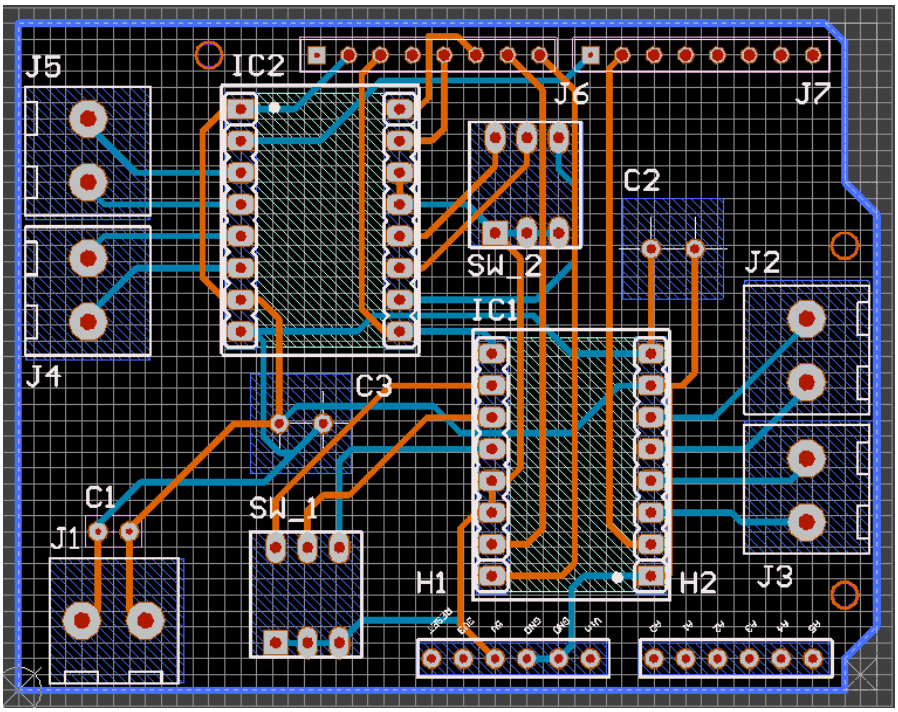


Fig. 17: PCB Layout and Routing

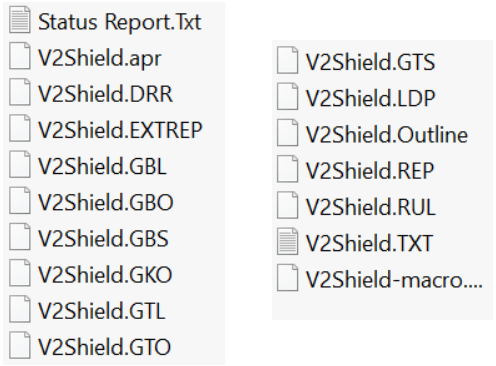


Fig. 18: Gerber Files for fabrication

The total cost of the fabrication and shipping was around 15 $ for 10 PCBs and, after 45 days, the final result arrived (**Fig. 19**). The quality was as good as expected and, after soldering all the components (**Fig. 21**), the PCB was tested and validated proving that the controllers worked as expected and the fabrication and design were not faulty.

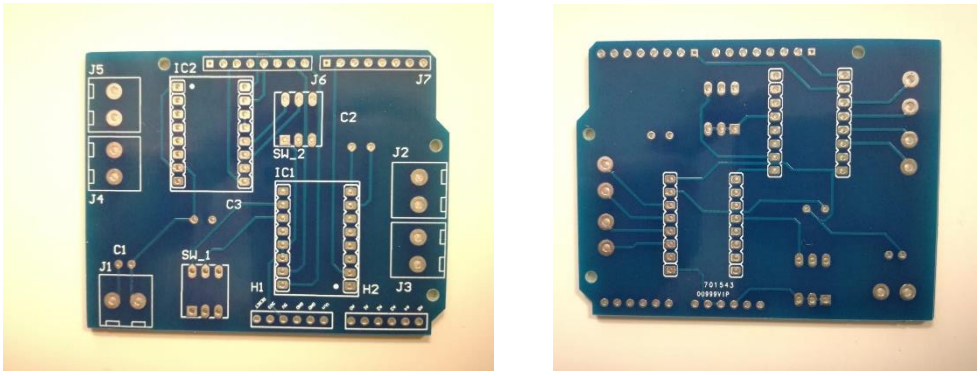


Fig. 19: *Fabricated Shield PCB*

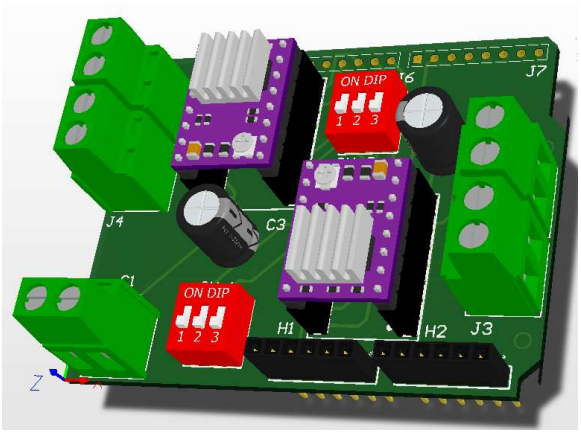


Fig. 20: *PCB 3D Model*

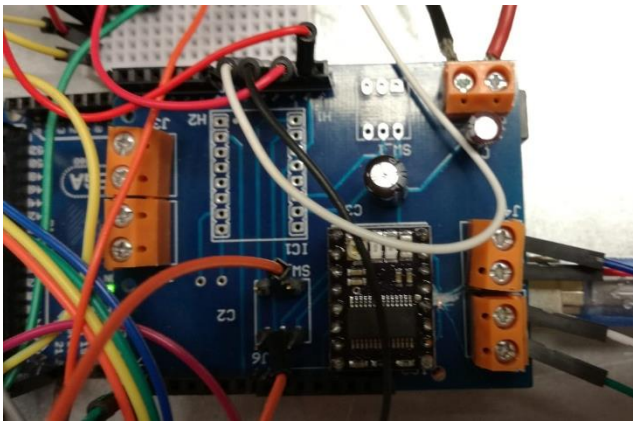


Fig. 21: *Implementation of PCB after Component Soldering*

3.3.3.- Accelerometer

As there is no way to control the motors in closed loop unless an encoder is used, the loop is closed through the angle reading of an accelerometer sensor that will move solidary to the body of the Glider. The main control idea is to move the motors a small number of steps each program period, and then check with the accelerometer measurements if the desired angle has been reached.

Considering the need of a reliable and versatile sensor, it was decided to use the GY-80 Module [13] (**Fig. 22**) which merges different sensors such as:

- 3 Axis Gyroscope
- 3 Axis Accelerometer
- 3 Axis Magnetometer
- Barometer + Thermometer

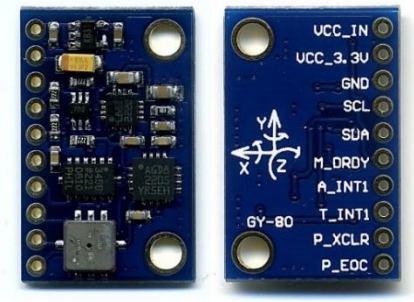


Fig. 22: *GY-80 Accel Module*

In order to transmit all of this information into the Arduino board, I2C communication is used. I2C is Master/Slave Bus where any number of devices can be connected, each of them have a unique address that will help identify which device is sending or receiving data. For this application, only the Accelerometer sensor will be used, whose address is 0x53, and only the pitch and roll axis are needed for the control.

In order to smooth out any peak that the sensor measurements may produce (**Fig. 24**[2]), a Kalman Filter (KF) [14][15] is applied before using the measurement as valid data for the closed loop control. As the code written in the Glider 1 for the KF works great, the same code will be used for this version (**Fig. 23**).

```
#include <math.h>
#include <MatrixMath.h>
#define PI 3.14159265358979f

//************************ Filter Variables **********************************
float x_angle=0;

float Q_angle  =  0.01; //0.001    //0.005
float Q_gyro   =  0.0;  //0.003  //0.0003
float R_angle  =  0.01;  //0.03     //0.008

float x_bias = 0;
float P_00 = 0, P_01 = 0, P_10 = 0, P_11 = 0;
float y, S;
float K_0, K_1;

float kalmanCalculate(float newAngle, float newRate, int looptime)
{
    float dt = float(looptime)/1000;
    x_angle += dt * (newRate - x_bias);
    P_00 +=  - dt * (P_10 + P_01) + Q_angle * dt;
    P_01 +=  - dt * P_11;
    P_10 +=  - dt * P_11;
    P_11 +=  + Q_gyro * dt;

    y = newAngle - x_angle;
    S = P_00 + R_angle;
    K_0 = P_00 / S;
    K_1 = P_10 / S;

    x_angle +=  K_0 * y;
    x_bias  +=  K_1 * y;
    P_00 -= K_0 * P_00;
    P_01 -= K_0 * P_01;
    P_10 -= K_1 * P_00;
    P_11 -= K_1 * P_01;

    return x_angle;
}
```

Fig. 23: *Kalman Filter code*

As the code also follows a modular fashion, this piece of code is included as a standalone module. The main aim of the KF is to deal with uncertainties, mainly white gaussian distributed noise, in the process as well as in the sensors of the system. To do so, KF match the estimated output of the observer (mathematical model of the system) with the real output, meaning that the estimated value of the internal variables will converge to the real value of this internal variable that cannot be measured directly.

[2] Busquets J., Busquets D.., Busquets J.V., "Combined Gas-Fluid Buoyancy System for Improved Attitude and Maneuverability Control for Application in Underwater Gliders", *IFAC-PapersOnLine*, 48-2 (2015) 281–287

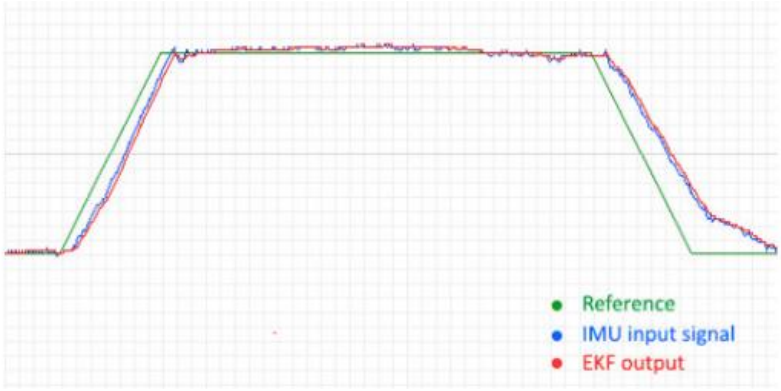


Fig. 24: EKF Filtering measurements obtained with Glider 1

The KF is an iterative process that works in two steps: the first one computes a prediction or "a priori Estimate" using the previous value of the estimated variable and the current input and the second step incorporates the measurements from sensors and updates the value of the "a posteriori estimate". The equations that perform this iterative process (**Fig. 25**[3]) are implemented into the KF code (**Fig. 23**).

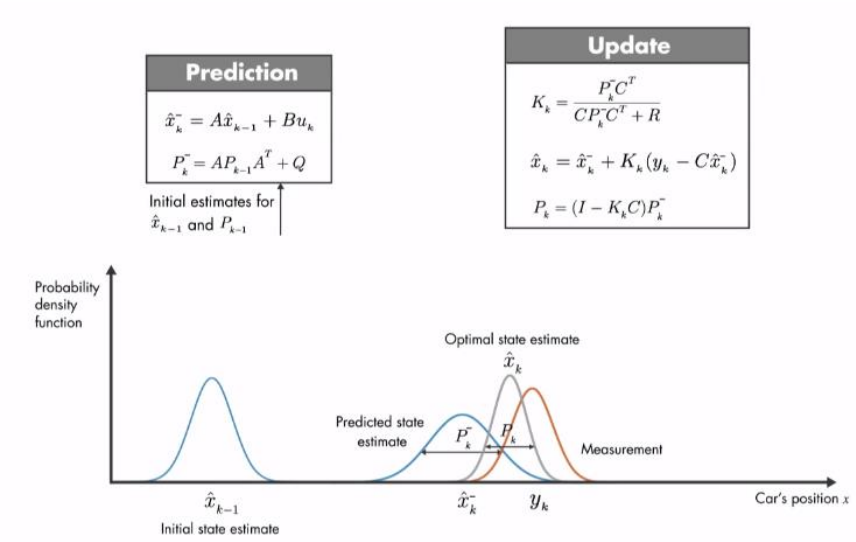


***Fig. 25:** Kalman Filter Equations*

Using this estimation algorithm and the measures from the sensors, the pitch and roll angles of the Glider are obtained and fed into the control of the stepper motors in order to keep the vehicle within user's specifications.

With the selection of the hardware components, which are common to both attitude control systems, well defined, the differences between the Pitch and Roll attitude control systems will be assessed together with explanations about the Arduino code written for that purpose.

[3] Anonymous, "Understanding Kalman Filters", Video and Webminar Series, (2017), https://www.mathworks.com/videos/series/understanding-kalman-filters.html, accessed January, 2018

3.4.- Pitch Axis

The pitch attitude control is one of the most vital systems for a Glider as its forward movement is produce by a sawtooth trajectory (**Fig. 26**[4]) and this control systems ensures that the angle of attack changes from negative to positive always within the user's specifications.

As it has been already specified, this change in the pitch angle is achieved by the movement of a mass in a longitudinal fashion along the body of the Glider. The mechanical system that translates the rotational force of the motor into longitudinal displacement was designed by another ETSID student, whose TFG is aimed at the mechanical design of these systems. The principle on which this translation works is an endless screw attached to the axis of the stepper via a pulley (**Fig. 27**).

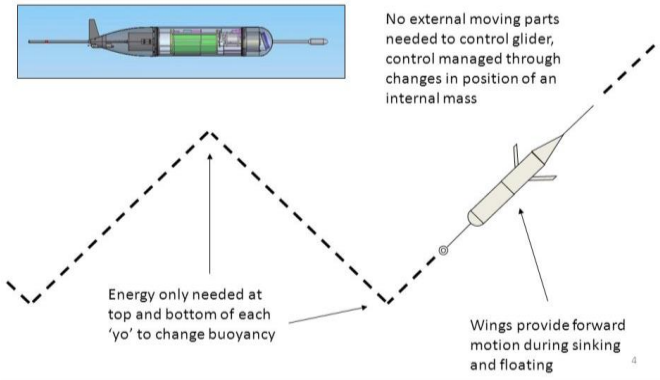


Fig. 26: Example of AUV's characteristic sawtooth movement

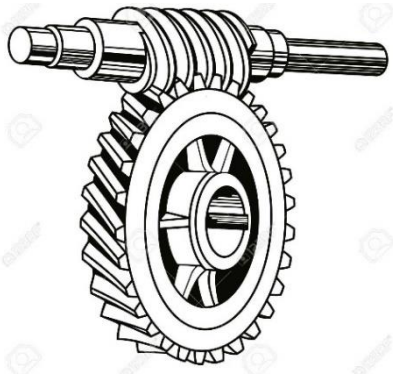


Fig. 27: Transmission principle used for the Glider

In order to execute a precise control on the mass' position, a transfer function (TF) is obtained relating how many motor steps produce a 1mm displacement of the mass using a trial and error method **(1)**. First, the motor was coded to move 200 steps but, due to the reduction produced by the planetary gearbox and the endless screw, the movement of the motor was too small to provide a relevant measure. Then, the motor was moved different number of steps until one complete revolution of the motor was achieved. Finally, the displacement of the moving tray was measured for one revolution of the motor and the TF was obtained **(2)**. The result of this testing yielded a ratio of 735 steps/revolution, meaning that the reduction ratio included on the motor datasheet was not correct and a new ratio was obtained **(3)**.

$$Test\ Results \quad \rightarrow \quad Ratio = 735\ \frac{Steps}{Rev} \qquad Tray\ displacement \cong 8\ mm \quad (\mathbf{1})$$

$$Translation\ TF = \frac{735}{8} = 91.88 \cong 92\ \frac{Steps}{mm} \quad (\mathbf{2})$$

$$Ratio\ w/o\ reduction = 200\ \frac{Step}{Rev} \rightarrow Reduction = \frac{735}{200} = 3.675 = 1 : 3.675\ (\mathbf{3})$$

[4] Annabel Chadbourne, "Coastal Glider Overview", Oceanology International, (2014), https://slideplayer.com/slide/2498065/, accessed March, 2018

3.4.1.- Motor and Electronics Housing

Due to the limited space inside the Glider, the housing of all the components should be studied carefully. Thus, every component of the housing has been designed in 3D and fabricated via a 3D printer. This design has been part of the final project of another colleague student, Alejandro [16]. All of the designs included belong to his work and are included here (**Fig. 28-29**[5]) as a way to illustrate the main structure of the attitude control system housing.

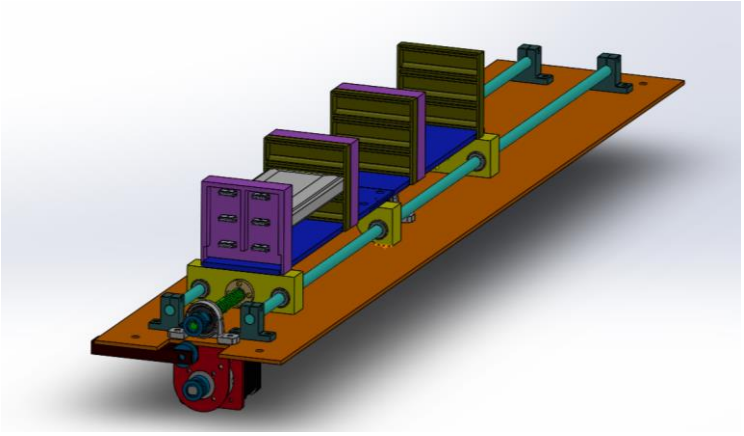


Fig. 28: *Attitude control system mechanical 3D design*

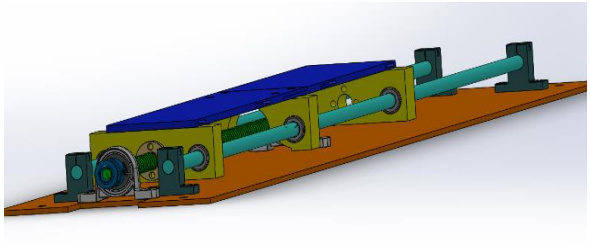


Fig. 29: *Detail of the inertial mass tray*

The main idea behind this clever design is to house the batteries that will power the stepper motor in the violet mountings seen in the images. This way the batteries act as a power source as well as dead weight for the attitude control system to work. All of the control systems (Arduino, sensors, electronics) will be housed on the back of the moving mass tray. The real implementation of this design together with the electronic systems is also included (**Fig. 30-31**).

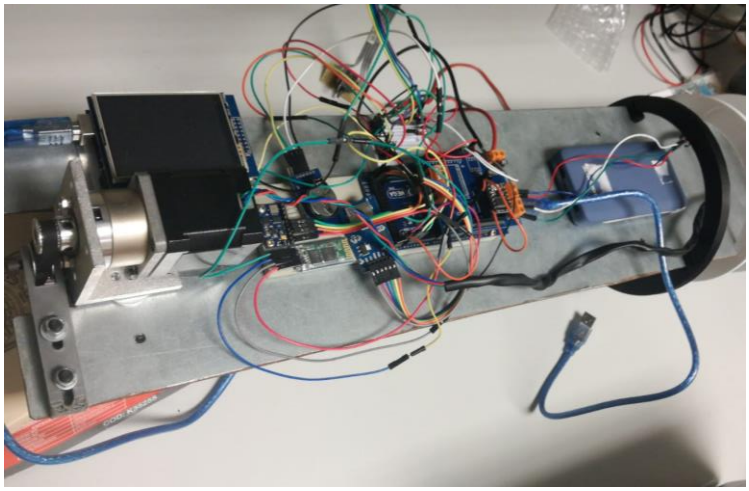


Fig. 30: *Bottom Side of the Moving Mass Tray*

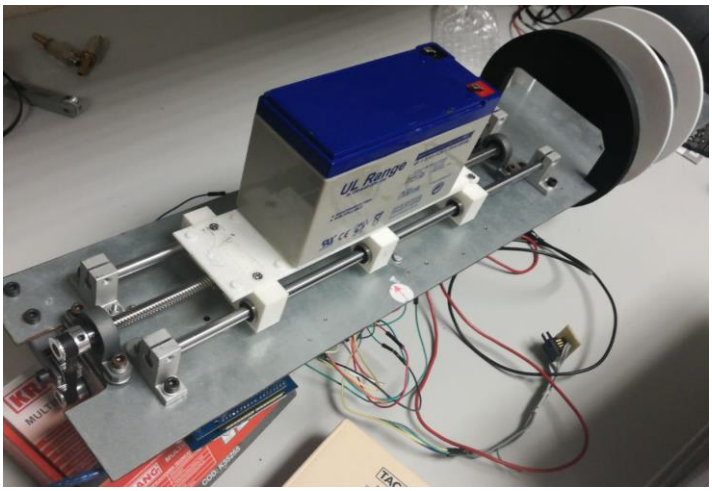


Fig. 31: *Upper Side of the Moving Mass Tray*

[5] Cebrián Abellán, A., "Sistema de desplazamiento de masas para el control de orientación de un Glider submarino", DISCA, (2018)

3.4.2.- Control Strategy

Once the preliminary measurements and the hardware design were performed, it was time to think of a way to control the stepper motor. Due to the fact that stepper motors are digital systems and there is no direct feedback from the motor, the following was decided:

- The motor will move 92 steps or 1 mm every cycle until the desired angle is achieved.
- Due to its digital nature, a simple Proportional (P) controller is implemented which will change the speed of rotation (RPM) of the motor depending on the angle difference between the desired angle and the measured one.
- The motor will be disabled when the desired angle or one of the ends of the moving tray are reached so as to reduce energy consumption. Mechanical brake is provided by the planetary gearbox and the endless screw transmission.

Using this strategy, the position of the mass is tracked at all time as the stepper motor will not lose any steps and will move the mass exactly 1mm each time, no matter the speed of rotation.

3.4.3.- Control Implementation

With the strategy well defined, the solution was coded in Arduino. The control code was implemented using the “Glider 1” code as a common ground. Thus, this section will be focused on the changes that allowed for the control of the stepper motor.

The first part of the process is to declare the stepper motor using the “BasicStepperDriver” library [17] together with the initialization parameters (**Fig. 32**) such as the Pins where the controller is connected to the Arduino (STEP, DIR, ENABLE), the number of steps per revolution (MOTOR_STEPS), the motor rotation speed (stepRPM) and the stepping mode (MICROSTEPS). This last parameter was selected to be ¼ of a step as it provides more precise movement when the speed of rotation is changing. Even though micro stepping provides slightly less torque, the noise and mechanical vibrations of the motor are greatly reduced. This is an important factor in order to keep the AUV stable. Electric consumption is not a decisive factor as the current draw in both full step and ¼ step mode is almost the same, being the latter the one with the least current draw (**Fig. 33**).

```
#include "BasicStepperDriver.h"
#define MAX_RAIL 120  //Length of the endless screw after testing with some margin (MAX = 275)
#define MOTOR_STEPS 200 //200 steps per revolution (1.8°)
#define MICROSTEPS 4 //1/4 steps (Best method for smoothness and torque)

//Pin configuration for motor
#define DIR 10 //(9)
#define STEP 11 //(8)
#define ENABLE 13

int stepRPM = 120; //Speed of Stepper Motor
BasicStepperDriver stepper(MOTOR_STEPS, DIR, STEP, ENABLE);  //Stepper driver declaration
```

Fig. 32: *Stepper Motor Initialization*

Together with the afore mentioned parameters used to declare the motor ('stepper'), an additional parameter regarding the physical environment of the control system is included. 'MAX_RAIL' refers to the real length of the endless screw mounting (**Fig. 28-29**). This is a constant that acts as a stop point for the motor's movement. Whenever the internal variable 'posRail' reaches the 'MAX_RAIL' value, the motor is powered off no matter the actual angle of the vehicle.

The next part deals with the declaration of the sensor (accelerometer) that closes the loop and the variables used for the motor's control (**Fig. 34**).

Stepping Mode	Electric consumption (A)
Full Step	1,25
¼ Step	0,8

Fig. 33: Motor electric consumption table

```
int RPM_a = 0;
long delta_tetha=0; //Angle difference for P control
unsigned long timer=0, delta_t=0; //delta time or how long it takes to execute data acquisition
int accx,accy,accz;   // integer Read from Accel

double setAngle, inAngle, outAngle; //Define Variables we'll be connecting to in PID
double angleOFF=0; //Angle offset for trimming procedure
//double kp = 2.0, ki = 0.0, kd = 0.0;
int margin = 10;
int numCycle=0, showAccel=0;
int posRail=60;
unsigned long timerChange = 0;
ADXL345 adxl; //variable adxl is an instance of the ADXL345 library
```

Fig. 34: *Sensor and control variables declaration*

The name and task inside the code of each of the variables are the following:

- **RPM_a:** OUTPUT variable of the control system (motor speed).
- **delta_tetha:** INPUT variable of the controller. It's the difference between the user defined angle (setAngle) and the vehicles' real angle (inAngle).
- **timer, delta_t:** variable used by the KF to smooth the accelerometer measurements as it is a time dependent algorithm.
- **accx, accy, accz:** variables used to store the readings of the accelerometer module in 3 axes.
- **setAngle:** USER defined REFERENCE for the control system.
- **angleOFF:** variable used for trimming the value provided by the accelerometer.
- **margin:** USER defined value to determine the angle values range that are within specifications.
- **posRail:** internal variable used to represent the position of the moving mass inside the vehicle.
- **adxl:** object declaration representing the accelerometer module.

To better understand the control system being coded, a block diagram is included (**Fig. 35**). Within this diagram, the indirect feedback (via Accelerometer & KF) can be seen and understood together with all of the system variables already explained.

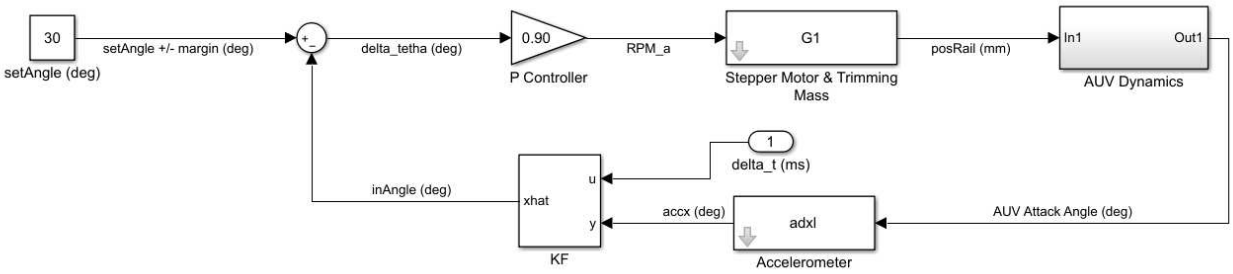


Fig. 35: AUV trimming control system diagram

As it has been already stated in section 2.4.2, a trial/error control strategy will be adopted meaning that the motor will move until the desired angle is reached. Since the vehicle's dynamics are way too difficult to model due to the ever-changing sea environment, the vehicle is included in the diagram (**Fig. 35**) as a blind subsystem block.

Once the variables and elements of the control system are well defined, the final step is to explain the controller algorithm (**Fig. 36**).

```
adxl.readAccel(&accx, &accy, &accz); //read the accelerometer values and store them in variables  x,y,z
delta_t = millis() - timer;                          // calculate time through loop i.e. acq. rate
timer = millis();                                      // reset timer

inAngle = kalmanCalculate(accx, 0.0, delta_t);
inAngle = (inAngle - angleOFF);  //Compensate observed offset in accel

//P speed controller
delta_tetha = (setAngle - inAngle);
delta_tetha = abs(delta_tetha);

RPM_a = delta_tetha*0.90; //In order to extend the range between min and max.

if (RPM_a >= 200){
  stepRPM = 200;
}else if (RPM_a <= 60){   //Limiting the minimum speed to ensure smooth motion.
  stepRPM = 60;
} else stepRPM = RPM_a;

//myPID.Compute();
stepper.begin(stepRPM, MICROSTEPS);
stepper.disable();
```

```
if (posRail<0){
  posRail=0;
  science.posRail = posRail;
}
if (posRail>MAX_RAIL){
  posRail=MAX_RAIL+1;
  science.posRail = posRail;
}
```

```
if (inAngle > setAngle + margin) {
  if (posRail<MAX_RAIL) {
          stepper.enable();
          delay(20);

          stepper.move(-MICROSTEPS*92); //the motor moves the tray exactly 1mm back or forth
          posRail++;

  }
}
else if (inAngle < setAngle - margin) {
  if (posRail>0) {
         stepper.enable();
         delay(20); //Delay introduced to avoid power bank automatic shutdown

         stepper.move(MICROSTEPS*92);

         posRail--;
   }
}
```

Fig. 36: Controller algorithm

The first part of the code (**Fig. 36**) deals with the data acquisition from the accelerometer and the KF. This reading, multiplied by the P controller gain, will provide the value for the motor's RPM. This gain is set to be 0.90 according to empirical measurements of the maximum angle difference between the user selected value and the real attack angle of the AUV.

Once the RPM value is obtained, the algorithm makes sure that this value is not higher than 200 RPM or lower than 60 RPM. Within this specification values, the motor is initialized with the calculated RPM value ("stepper.begin"). Finally, the algorithm checks if the input angle coming from the KF is within user's specifications. Depending on whether the real angle is above or below the user specified angle, the motor will perform 92 steps (1mm mass displacement) in clockwise or counter clockwise direction. Otherwise, the motor will remain disabled and it will not move. The middle part of the code ensures that the mass will not move further than the length of the rails according to the variable "*posRail*", which is updated every time the motor rotates.

All the decision process of the controller algorithm is synthesized in a flow chart (**Fig. 37**).

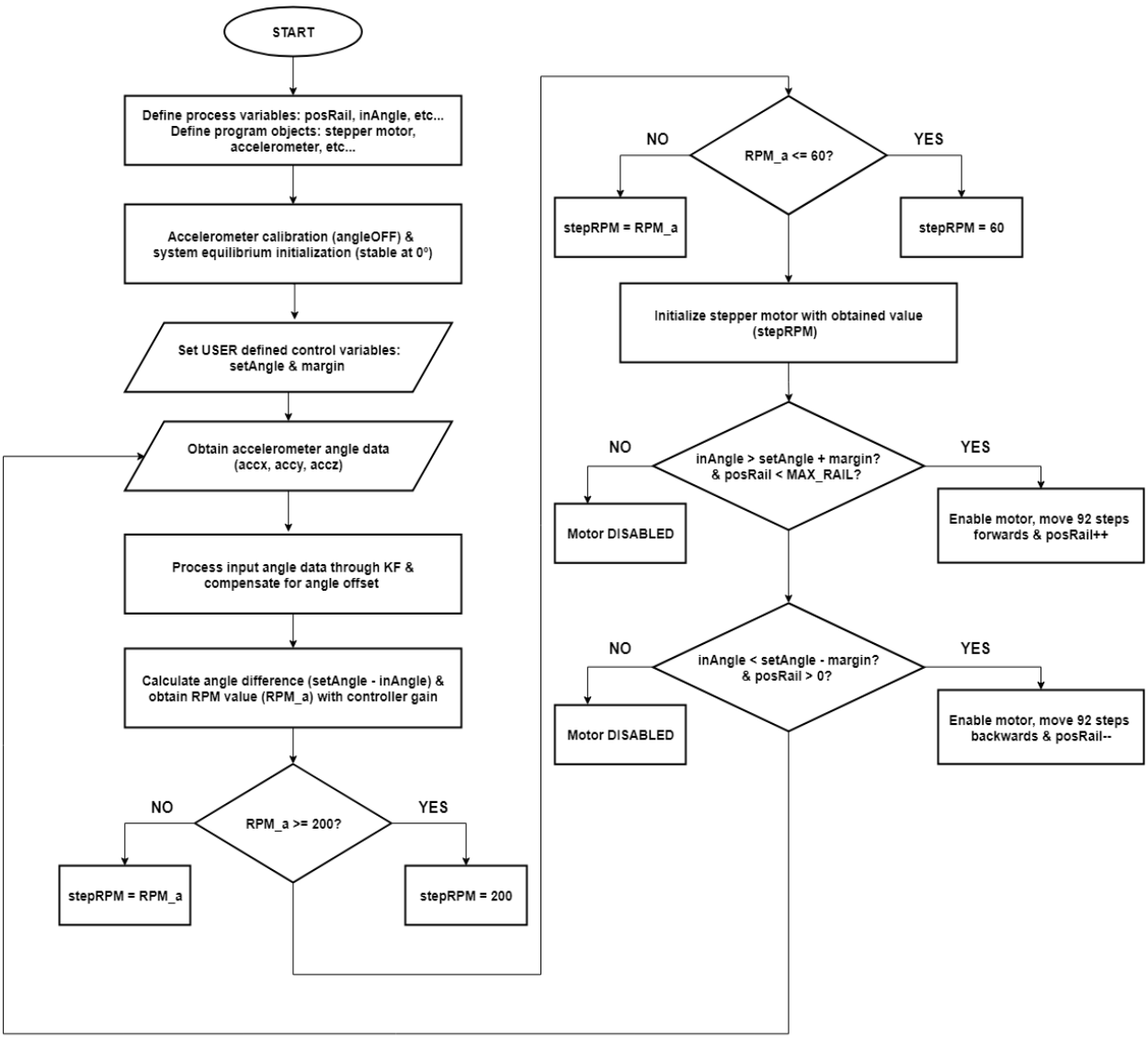


Fig. 37: *Attitude control system flow chart*

The loop seen in the flow chart (**Fig. 37)** repeats until the system is powered off or stopped manually via Bluetooth. Communication systems will be assessed in the next chapter. The final part of this algorithm performs the sawtooth movement required for the AUV to move forward. This is achieved by counting the number of loop cycles (numCycle) and, when this variable reaches a user defined variable (timerChange), the 'setAngle' is changed from positive to negative and vice versa (**Fig. 38**).

```
if (numCycle > timerChange && timerChange>1000) {
    numCycle=0;
    setAngle = -1 * setAngle;
    miSerial.print(" changed angle: ");
    miSerial.println(setAngle);
}
```

Fig. 38: *Sawtooth angle change algorithm*

3.5.- Roll Axis

In order to change the course of the 'Glider', a roll axis trimming system is also included. As said before, this system consists of a rotational mass whose displacement is translated into a change in the roll angle of the vehicle. This system, together with the variable buoyancy systems placed on the wings of the AUV and the pitch trimming system, will allow the 'Glider' to turn in any direction in order to perform corrections on the course or avoid obstacles.

Despite being a rotational system, the control strategy is the same as in the pitch axis system. As stated before, the same hardware will be used for both systems and every similarity between them has already been discussed. For this reason, this section will focus only on the differences, namely the system TF and physical implementation.

3.5.1.- Motor and Mass Housing

Before explaining how the TF was obtained, it is important to show how the motor is connected to the rotational mass. As in the previous section, the 3D design implemented (**Fig. 39**[6]) was designed by my college Alejandro. This time, the stepper motor is also connected to an endless screw which is then mashed into a round gear that moves solidary to the rotational mass. This whole system is integrated with the previously explained pitch trimming system housing. By the time this work is performed, the roll axis housing is not mounted physically, and no picture is available.

[6] Cebrián Abellán, A., "Sistema de desplazamiento de masas para el control de orientación de un Glider submarino", DISCA, (2018)

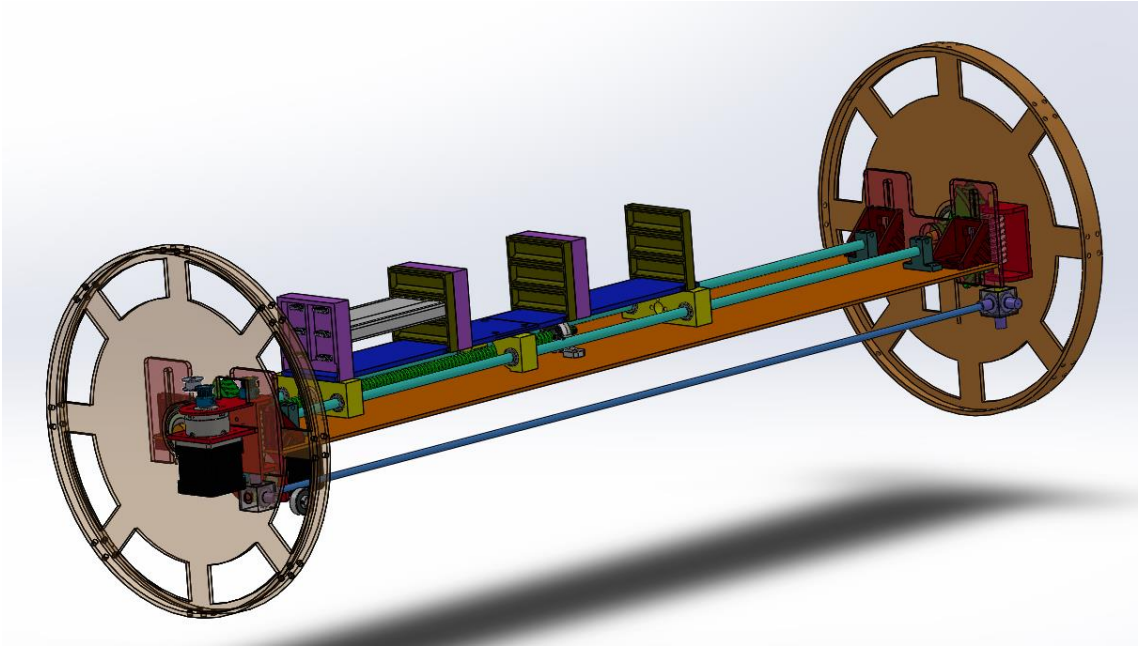


Fig. 39: *Complete trimming system 3D design*

3.5.2.- Transfer Function Obtention

In this system, the stepper motor is also joined with the rotating mass by an endless screw. However, the screw is directly mashed with a circular gear connected to the rotating mass by its axis. In order to obtain the TF, the radius and the distance between two consecutive teeth of the gear (gear step) are measured (**4**). Knowing that one full rotation of the endless screw produces a full step linear displacement in the gear, the TF relating the number of steps needed to rotate the gear by one degree is obtained (**5.1, 5.2**).

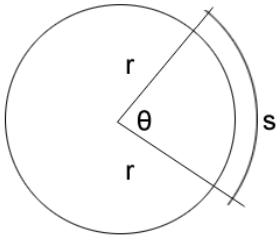


Fig. 40: *Circular gear simplification for calculus*

$$Teeth\ Arc\ Length = S = 6.7\ mm$$

$$Gear\ Radius = R = 46.2\ mm \qquad Arc\ Angle = \theta \quad (\mathbf{4})$$

$$S = R \cdot \theta \rightarrow \theta = \frac{S}{R} = \frac{6.7}{46.2}\ rad$$

$$\theta(°) = \frac{360° \cdot {}^{6.7}/_{46.2}}{2 \cdot \pi} = 8.31\,° \qquad (\mathbf{5.1})$$

$$Steps\ per\ Rotation\ with\ Reduction = 735 \quad \rightarrow \quad TF = \frac{735}{8.31} \cong 88\ \frac{Steps}{°} \qquad (\mathbf{5.2})$$

4.- Communications systems

4.1.- Introduction

Every vehicle needs a proper communication system in order to report back mission results, provide real time feedback of its sensors or receive commands from the user. In order to ensure that the Glider AUV is properly communicated with its environment and the final user, the following communication systems have been developed and implemented:

a) **Bluetooth communication system:** used to stablish a link between the user and the vehicle providing real time sensor date and enabling the user to send different commands to the glider.
b) **SPI communication system:** used to read/write data from different devices such as the SD card storage module.
c) **I2C communication system:** it is an internal bus that connects different devices in the vehicle including communication between different Arduino boards.

On a higher level, the Glider is also fitted with GPS and Wi-Fi communication systems for long distance trips. However, the scope of this work is limited to the low range communication systems displayed above. This section will focus on all three communication procedures including code explanations and physical implementation.

4.2.- Previous work

Communication systems were taken into account on the previous version of the Glider code. However, they solely focused on Bluetooth communication. For this reason, the commands already designed for the control of the vehicle (**Fig. 41**) were kept and modified to fit the new motors' specifications and a different Bluetooth module was used. SPI and I2C communication were not very developed as there were fewer devices in the previous versions of the Glider.

```
ch = miSerial.read();
miSerial.print("Leido ");
miSerial.println(ch);
switch (ch) {
   case '1':relay1on(); break;
   case '2':relay2on(); break;
   case 'S':
   case 's':relayStop(); break;
   case 'I':relayInterval++; break;
   case 'i':relayInterval--; break;
   case 'D':relayDutyOn++; break;
   case 'd':relayDutyOn--; break;
   case 'C':relayCompensate++; break;
   case 'c':relayCompensate--; break;
   case 'A':setAngle++; break;
   case 'a':setAngle--; break;
   case 'M':margin++; break;
   case 'm':margin--; break;
   case 'T':setAngle = 25; break;
   case 't':setAngle = -25; break;
   case 'H':
   case 'h':setAngle = 0; break;
   case 'V':showAccel = 1; break;
   case 'v':showAccel = 0; break;
   case 'X':timerChange+=1200; numCycle=0;
   break;
   case 'x':timerChange-=1200; numCycle=0;
   break;
   case 'Q':storeYes=1; numCycle=0;   break;
   case 'q':storeYes=0; numCycle=0; break;
   case 'W':showrecordAngle();break;
   break;

}
```

Fig. 41: *Glider 1 Bluetooth receive event with commands*

4.3.- Bluetooth (BT) communication system

4.3.1.- Hardware Design

The first step in designing the BT communication system is the hardware selection, this is, choosing the right BT module for the work. The two most reliable and used BT modules in the market are the HC-05 and the HC-06 [18] (**Fig. 42**). This time, the HC-06 BT module was selected according to the following:

- **Simplicity:** HC-06 module has only 4 pins compared to the 6 pins of the HC-05.
- **Duty:** As the BT module works only as a slave in the communication procedure, the master-slave feature of the HC-05 did not add any value to the decision.
- **Price:** HC-05 prices go as low as 4 € whereas HC-06 prices reach a low of 2.5 €
- **Space:** Given the reduced functionality of the HC-06 and the lesser number of pins, the design is more compact allowing for more space inside the Glider.

Fig. 42: *HC-06 Bluetooth Module*

Once the device is selected, the wiring must be assessed (**Fig. 43 & 44**). Out of the 4 pins, 2 of them are connected to Vcc and GND while the other two are the data transmission pin (TX) and the data reception pin (RX). These last pins are connected to one of the four Arduino MEGA serial ports swapping the RX on the BT module with the TX pin on the Arduino and vice versa.

Due to several difficulties in configuring any serial port of the Arduino MEGA other than the default (which is also used for USB communication with the board), Serial Port 0 was selected in order to perform the BT communication of the Glider.

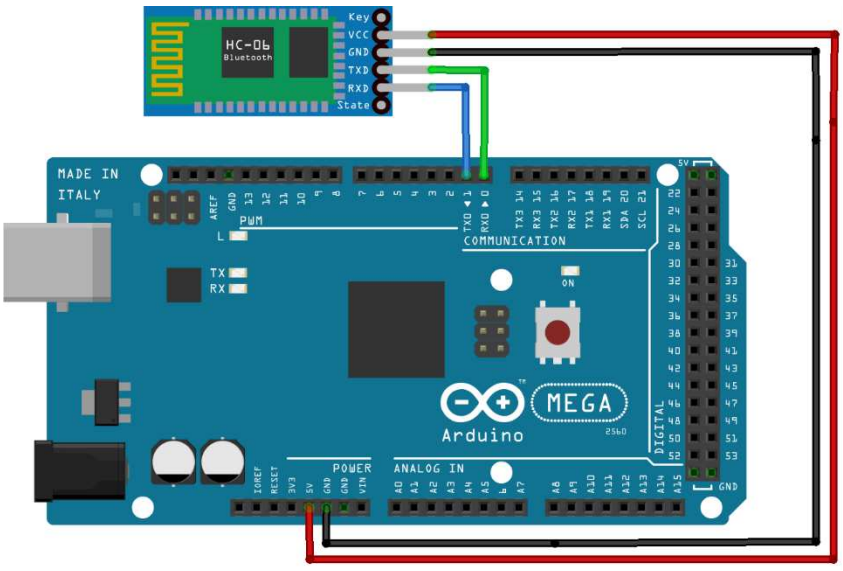


Fig. 43: *Graphic representation of HC-06 module wiring*

PIN CONNECTION TABLE	
HC-06 Module	Arduino MEGA Board
VCC	5V
GND	GND
TXD	PIN 0 (RX 0)
RXD	PIN 1 (TX 0)

Fig. 44: *HC-06 Module Wiring Table*

4.3.2.- BT Module Configuration and Code

The first time one of these modules are powered up, you need to select the BT device name, the pairing password and the serial connection speed. This is performed by sending a specially formatted messages (**Fig. 45**), called AT commands [19], that can be understood by the BT module when sent over the Arduino Serial Port. This configuration is performed only once as the device remembers all the parameters even after alimentation is withdrawn. For the Glider 3, the configuration was as follows:

- **Device Name:** Glider3
- **Pairing Password:** 1234
- **Connection Speed:** 9600 baud

```
#include <SoftwareSerial.h>
#define miSerial Serial // OJO poner Serial3 para blue (Bluetooth now
connected to Serial Port 0, same as USB)

void initBlue() {
      command("AT",2);// response: OK
      command("AT+VERSION",12);// response: OKlinvorV1.5
      command("AT+NAMEGlider3",9);//response: OKsetname
      command("AT+BAUD4",8);//response: OK9600
      command("AT+PIN1234",1);//response:
}

void startControl()
{
      //Communication
      miSerial.begin(9600);

      // Blue
      //initBlue();  //Uncomment only for configuration of new BT module
```

Fig. 45: *HC-06 Module Configuration Code*

As seen (**Fig. 45**), the first step in configuring the BT module is to stablish the Serial Port where the communication is happening (Port 0 in this case). Then, a function that sends the configuration AT command to the module is created ('*initBlue*'). Finally, the same communication speed selected for the BT module is selected for the Arduino board port to ensure proper function of the system. If a new BT module is to be configured, the next line of code will be uncommented, and the configuration function shall execute.

The next piece of code that is related to the BT communication system is the command table that the AUV is able to interpret and perform. The code structure for this table is the same as the one shown in point 3.2 (**Fig. 41**). However, some commands have been modified in order to properly control the stepper motors and some other new commands have been introduced (**Fig. 46**).

Command Char	Action	Command Char	Action
1	Stepper motor moves 10 mm backwards	**m**	Decrease angle margin by one
2	Stepper motor moves 10 mm forwards	**T**	Set target angle to 25°
3	Stepper motor moves 5 mm backwards	**t**	Set target angle to - 25°
4	Stepper motor moves 5 mm forwards	**H**	Compensate accelerometer offset
5	Stepper motor moves 3 mm backwards	**h**	Set target angle to 0°
6	Stepper motor moves 3 mm forwards	**V**	Show current real angle value
S	Set variable 'posRail' to 60 mm (midpoint)	**v**	Hide current real angle value
s	Stepper motor is stopped	**X**	Increase time for angle value swap by 1.200 code cycles
I	Increase motor RPM by one	**X**	Decrease time for angle value swap by 1.200 code cycles
i	Decrease motor RPM by one	**Q**	Enable data storage
A	Increase target angle by one	**q**	Disable data storage
a	Decrease target angle by one	**W**	Show stored angle value
M	Increase angle margin by one		

Fig. 46: *Bluetooth Command Table*

```
ch = (char)miSerial.read();
switch (ch) {
      case '1':
            stepper.enable();
            delay(20);
            stepper.move(MICROSTEPS*910);
            posRail = posRail - 10;
            break;
      case '2':
            stepper.enable();
            delay(20);
            stepper.move(-MICROSTEPS*910);
            posRail = posRail + 10;
            break;

      case 'S':posRail = 60; break;
      case 's':stepper.stop();
               angleOFF = inAngle;
               setAngle = 0;
               break;
      case 'I':stepRPM++; break;
      case 'i':stepRPM--; break;
```

```
      case 'A':setAngle++;
               miSerial.println(setAngle);
               miSerial.print(" ");
               miSerial.print(margin);
               break;
      case 'a':setAngle--;
               miSerial.println(setAngle);
               miSerial.print(" ");
               miSerial.print(margin);
               break;

      case 'H':angleOFF = inAngle; break;
      case 'h':setAngle = 0; break;
      case 'V':showAccel = 1; break;
      case 'v':showAccel = 0; break;
      case 'X':timerChange+=1200; numCycle=0;
               break;
      case 'x':timerChange-=1200; numCycle=0;
               break;
      case 'Q':storeYes=1; numCycle=0; break;
      case 'q':storeYes=0; numCycle=0; break;
      case 'W':showrecordAngle();break;
     break;
}
```

Fig. 47: *Example of Command Code Structure*

These commands (**Fig. 46 & 47**) allow the user to change the working parameters of the Glider on the fly, retrieve real time information, compensate the offset of the accelerometer measurements when the vehicle is powered and place the moving mass in the centre of the mass tray before the Glider starts to move. Using these commands, the user will perform the following sequence in order to set the initial conditions for the flight:

1. Place the Glider on the surface of the water as horizontal as possible.
2. Send command 'H' to set the current angle as 0° by compensating the initial offset.
3. Send command 'h' to set the target angle as 0°. This will prevent the motor from moving automatically.
4. Once the motor is stopped, the used is enabled to move the motor at will until the moving mass is in the centre of the moving tray using commands '1' through '6'.
5. Send command 'S' to tell the microcontroller program that the mass is centred ('posRail' = 60)
6. Set the desired target angle and margin using commands 'A', 'a', 'M', 'm', 'T' and 't'.

4.3.3.- Real Time Data and Android Control App

As the Glider will not be connected to a computer while underwater, the commands will be sent through a smartphone. Sending the command as letters and number though a serial monitor App can lead to confusion and mistakes. Thus, a specific App has been developed, providing the user with a GUI that allows sending commands and receiving real time data easy and visual. This APP has been developed using the online editing site "AppInventor2" from the MIT [20].

Due to buffer limitations on the serial bus, real time data (RTD) cannot be sent over to the mobile APP every microcontroller cycle. Thus, RTD is sent every 20 cycles and whenever the stepper motor moves the mass (**Fig. 48**). Besides, whenever a parameter is modified through a command (**Fig. 46**), the new value for said parameter is also sent through the serial bus (case 'A' and 'a' in **Fig. 47**).

```
if (inAngle > setAngle + margin) {
    if (posRail<MAX_RAIL) {
            stepper.enable();
            delay(20);
            stepper.move(-MICROSTEPS*92);
            posRail++;

           miSerial.print(inAngle);
            miSerial.print(" ");
           miSerial.print(posRail);
           miSerial.print(" ");
            miSerial.print(depth);
      }
}
```

```
void testControl()//Executed every 20
cycles
{

  miSerial.print(inAngle);
  miSerial.print(" ");
  miSerial.print(posRail);
  miSerial.print(" ");
  miSerial.print(depth);
}
```

Fig. 48: Real Time Data Sending Through Serial Bus Procedure

Having defined how the data is sent and received through the Serial BT channel, the mobile APP is developed. This APP consist of four different screens. The first one (**Fig. 49**) is the main menu, where RTD from the glider is displayed and the user is able to access the different sub-menus by clicking on the 4 buttons at the bottom of the screen. On top of this screen, the user can select the BT device to be paired with the APP by clicking on the "DEVICES" button or end this pairing by clicking on the button right next to it.

The second screen (**Fig. 50**) is displayed when the button "MASS TRIM CONTROL" is pressed. This screen displays the several BT commands available to move the motor at will in order to trim its position as explained in part 3.3.2. By clicking on any of these buttons, the APP will send the corresponding BT command to the microcontroller. This provides a visual way of sending commands without having to remember all of the numbers and letters (**Fig. 46**).

The following screens allow the user to set new navigation parameters for the Glider. The third one (**Fig. 51**) is displayed whenever the button "VALUE SETTING I" is pressed allowing the user to increase or decrease several parameter values by one. The last screen (**Fig. 52**) is shown when the button "VALUE SETTING II" is pressed and includes several commands that provide a predefined value to some navigation parameters.

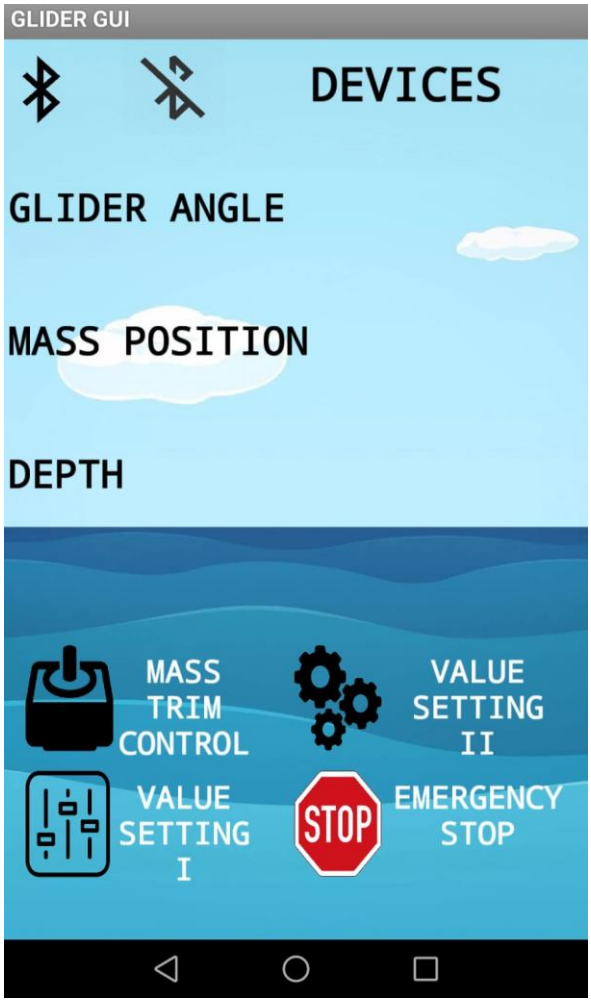


Fig. 49: *APP Main Menu*

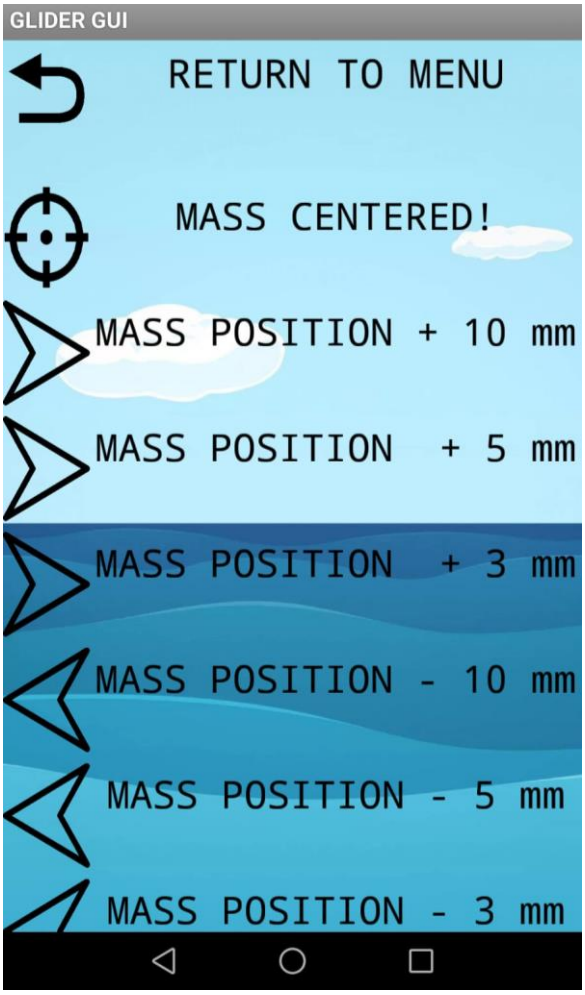


Fig. 50: *APP Trim Control Menu*

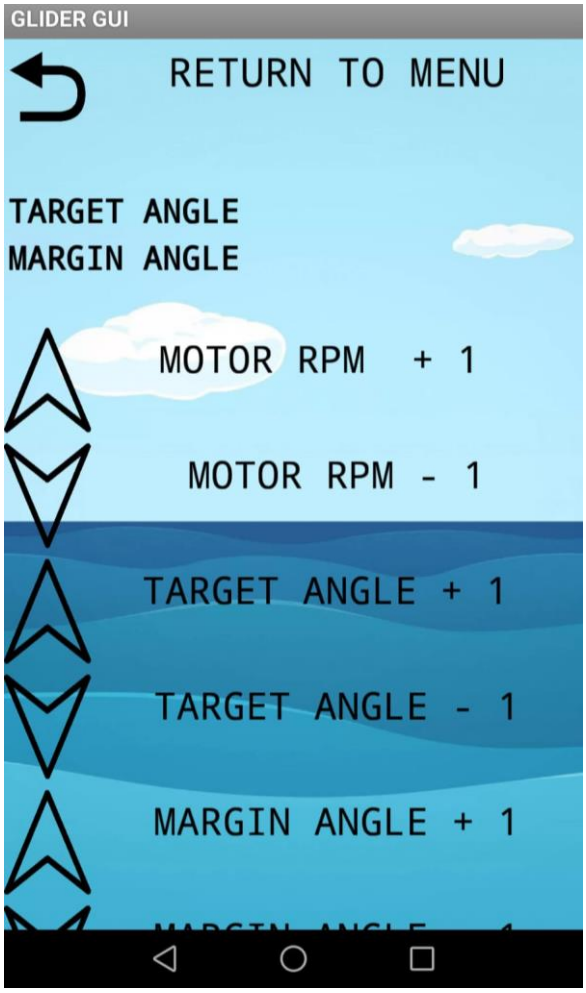


Fig. 51: APP Parameter Setting Menu I

Fig. 52: APP Parameter Setting Menu II

App Inventor 2 uses a visual programming language, developed by the MIT, called "Scratch"[21]. This language is easy to learn and use even with basic programming or APP creation knowledge. The design process starts with the visual creation of every screen displayed above and all of the objects they contain (buttons, labels, lists, etc…). Once the necessary objects are laid onto the screen, App Inventor will provide you with the different functions associated with each of these objects in order to start programming.

The algorithm is created by attaching different function blocks in order to obtain the desired result each time an event happens. For example, RTD display is triggered every APP clock cycle (**Fig. 53**) as long as the main menu screen (**Fig. 49**) is showing. Due to the fact that variables cannot be sent over a serial bus, the three different RTD values are separated by a space (**Fig. 48**) when sent. This space then helps the APP to differentiate between the three values and assign them to their corresponding label. Same thing happens for the parameters value when changed.

Having covered the reception of RTD by the APP, the next step is to send the right command when a command button is pressed on the app. The BT command character to be sent by each button is hardcoded and unique. Before sending the command and after the button is pressed, the program checks that BT communication is properly stablished (**Fig. 54**).

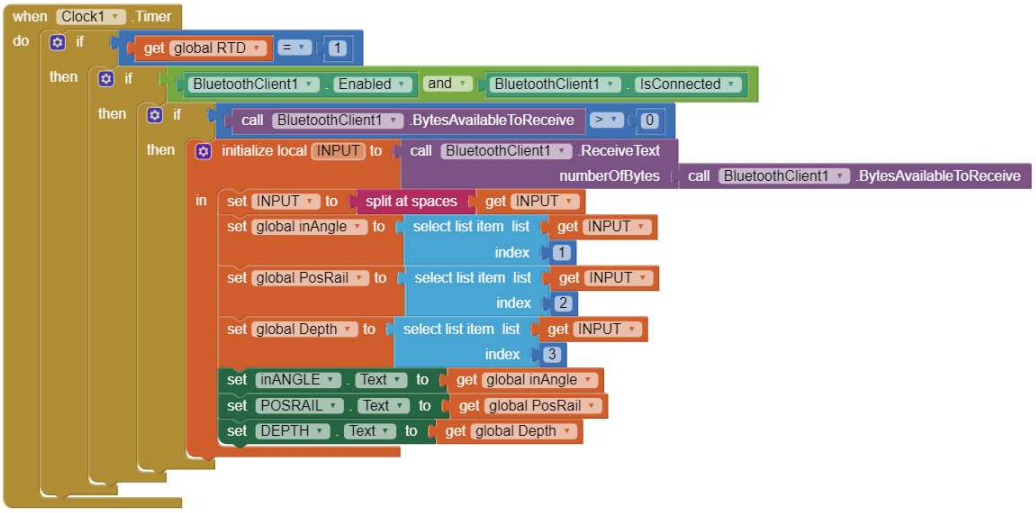


Fig. 53: *RTD Reception and Display Block Algorithm*

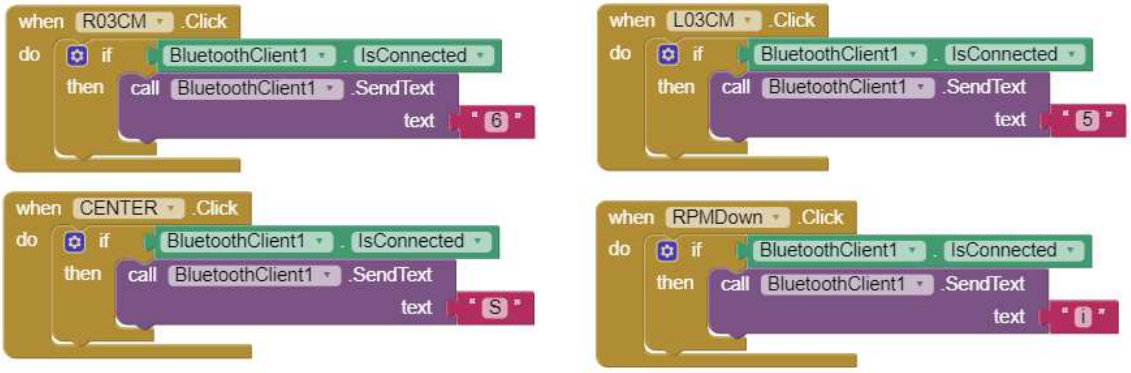


Fig. 54: *BT Command Sending Block Algorithm Example*

4.4.- SPI Communication System

4.4.1.- Introduction

SPI stands for Serial Peripheral Interface and it is a commonly used method to connect several devices to one single microcontroller [22]. SPI is a synchronous communication procedure, as opposed to the usual Serial Bus explained before. This means that, alongside the data lines, there is a clock signal connecting both the device and the microcontroller which rules the communication process. This clock signal ensures that every piece of information is sent or received on a falling or rising clock signal edge (**Fig. 55**[7]).

Being synchronous, SPI communication allows for a faster and larger data exchange while ensuring that no information is lost in the process. For these reasons, SPI communication is used in this project to connect the SD Card module to the Arduino Mega as a navigation data log.

[7] Mike Grusin, "Serial Peripheral Interface (SPI)", Tutorials, https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all, accessed November, 2018

SPI is also based on a Master/Slave system that allows to connect several slaves (devices/modules) to a single Master (microcontroller). To do so, SPI communication bus includes one extra line used to select the slave to exchange information with (**Fig. 56**).

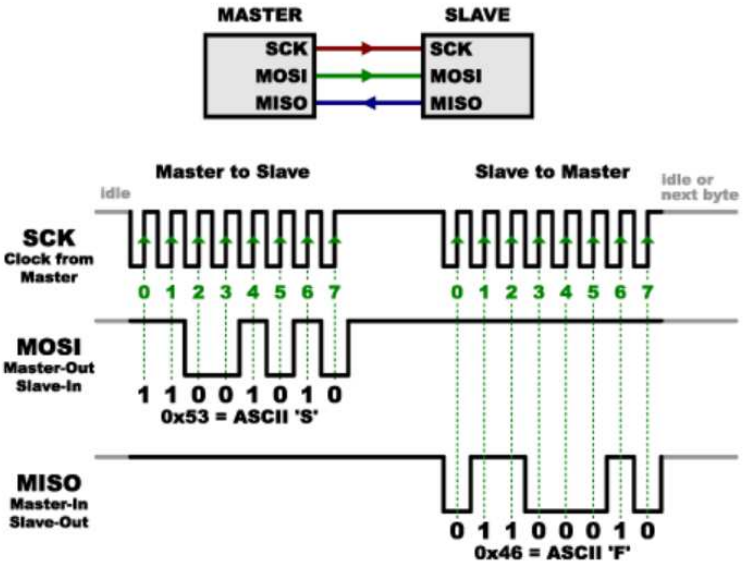


Fig. 55: SPI Communication Example

Acronym	Description
SCK	Seria Clock
MOSI	Master Out/ Slave In
MISO	Master In/Slave Out
SS	Slave Select

Fig. 56: SPI Bus Lines Description

4.4.2.- Hardware Design

As stated before, SPI communication is only used to log navigation data into a SD card that will be used to analyse the retrieved data via Excel. Further explanations on this procedure are given on Part 6 of this document.

The SD card module used in this project is a generic SD Card Adapter (**Fig. 57**) that is compatible with Micro SDHC cards. This type of cards can store larger data volumes at a higher transfer speed, allowing the data log files to be written many times over the execution of one program cycle. This generic module is connected to the Arduino MEGA with a SPI bus.

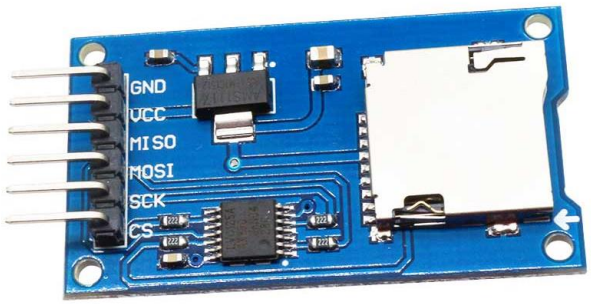
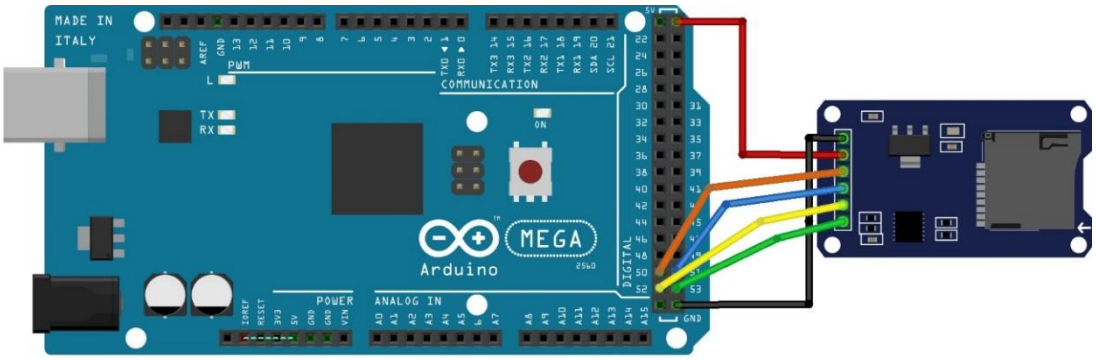


Fig. 57: SD Card Adapter

In this part, the focus will be the wiring of the SD Card module (**Fig. 58**) to the microcontroller in order to stablish proper communication and the Arduino configuration code (**Fig. 59**).



PIN CONNECTION TABLE	
SD Card Module	Arduino MEGA Board
VCC	5V
GND	GND
MISO	PIN 50
MOSI	PIN 51
SCK	PIN 52
CS	PIN 53

Fig. 58: *SD Card Adapter Module Wiring Schematics and Table*

To configure the SPI communication for the SD card adapter, the Arduino library called “SD” has been used [23]. The Arduino MEGA pins selected for the project are the ones defined as default by this library (**Fig. 59**). Besides, the “SD” library provides the user with several file treatment functions such as open file, save, close, create, etc… that makes it easier to create the navigation database.

```
#include <SD.h>
#include <Arduino.h>

//SPI settings
//MOSI,MISO,SCLK set as default
int CS_pin = 53;
int SDin = 1;
```

Fig. 59: *SPI Bus Configuration Code*

4.5.- I2C Communication System

4.5.1.- Introduction

I2C communication is another type of master/slave synchronous serial bus developed by Philips in the 1980s in order to control several TV chips [24]. The main difference between I2C and SPI communication is that I2C uses only two lines (**Fig. 60**[8]), one for the clock signal (SCL) and another for the data (SDA). This means that both the master and the slave can send or receive data through the same line. To do so, each device connected to an I2C bus has its unique hexadecimal address inside the bus.

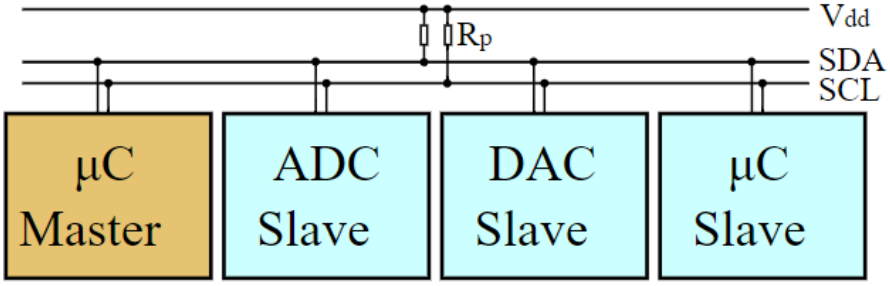


Fig. 60: *Example of I2C wiring*

As every device connected to the bus can be either a slave or a master, this differentiation is made programmatically inside the microcontroller code. To stablish communication with the right device inside an I2C bus, the master sends the device address he wants to communicate with through the bus. Then, every device check if the address requested matches its own address. If so, said slave sends the master an acknowledge bit indicating that the device is ready for the sending/reception of data [25].

The main advantages of I2C communications are the following [25]:

- Simple and powerful communication interface using only two bus lines.
- Devices can be both master or slave.
- The address space allows for the connection of up to 128 different devices.
- Up to 400 kHz data transfer speed

4.5.2.- Hardware Design

As I2C communication system is fast, reliable and allows for the connection of many devices, a great number of Arduino modules use I2C as default. In the case of this project, the following devices are connected to the bus:

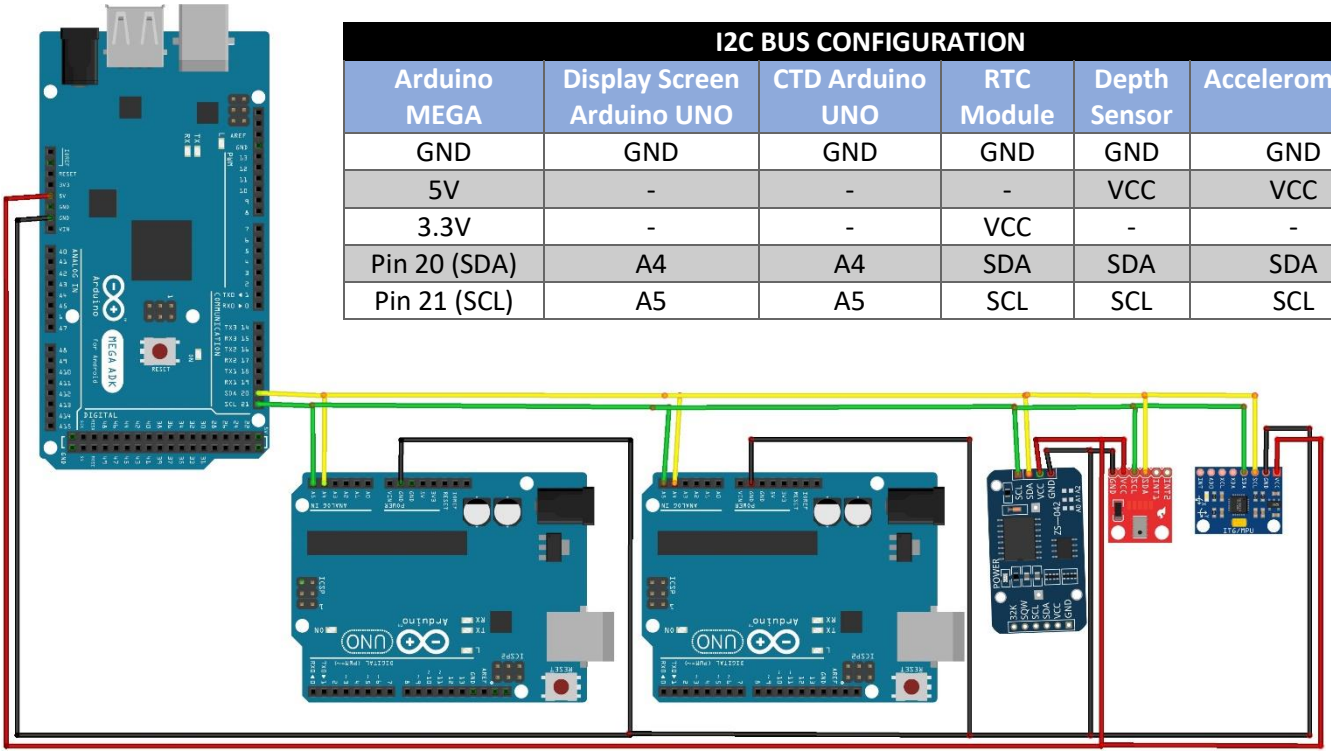
- Accelerometer **(Part 2)**
- Real Time Clock (RTC) **(Part 6)**

[8] Anonymous, "I2C", *Aprendiendo Arduino*, https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/, accessed January, 2019

- Depth Sensor **(Part 5)**
- Display Screen Arduino Board **(Part 4)**
- CTD Arduino Board **(Part 5)**

This part will not focus on each of the devices but on the wiring and addressing of the I2C bus. Three of these devices are Arduino modules which have their own specific library and address. Thus, the I2C communication protocol is already assessed and there is no need for any bus configuration. However, there are two Arduino UNO boards that will work as slaves under the commands of the Arduino MEGA board. The bus configuration for each of these boards will be explained in the corresponding part highlighted above.

Once the number of devices to be connected and their corresponding addresses (**Fig. 62**) are defined, the different modules and boards are connected to the I2C bus (**Fig. 61**).



I2C BUS CONFIGURATION					
Arduino MEGA	Display Screen Arduino UNO	CTD Arduino UNO	RTC Module	Depth Sensor	Accelerometer
GND	GND	GND	GND	GND	GND
5V	-	-	-	VCC	VCC
3.3V	-	-	VCC	-	-
Pin 20 (SDA)	A4	A4	SDA	SDA	SDA
Pin 21 (SCL)	A5	A5	SCL	SCL	SCL

Fig. 61: *I2C Bus wiring configuration for the Glider*

I2C BUS ADDRESSES	
DEVICE	ADDRESS
Arduino MEGA Board	0x07
Display Screen Arduino UNO	0x08
CTD Arduino UNO	0x09
RTC Module	0x56
Depth Sensor	0x40
Accelerometer	0x53

Fig. 62: *I2C Bus unique device addresses*

5.- Vehicle Status Display Screen

5.1.- Introduction

In this part, the configuration, design and physical implementation of a display screen GUI for the glider will be assessed. The main idea behind the addition of a screen is to have real time data shown to the scuba divers or to the operator of the Glider while underwater. This feature will be especially important in the testing stage of the AUV where it must be ensured that all the different systems work properly on site. After this stage, the screen will come in handy as an auxiliary real time data log in case the developed APP fails. To fully understand the whole design and implementation of this device, this part will be divided as follows:

- Hardware selection
- GUI design code on Arduino UNO
- Variable configuration of I2C channel between Arduino UNO and MEGA

5.2.- Hardware selection

The first hardware component to be selected was the display screen. As with every other part of this project, the key criteria for component selection were lightness, low cost and modularity. Although resolution and brightness are also important features to consider, they are not as relevant in this case as the screen does not need to display any complicated graphics or images, just letters and real time data numbers. Taking these filters into consideration, the selected screen was a 2.8'' LCD Touch Screen Arduino Shield [26] (**Fig. 63**) from a low-cost manufacturer called "ELEGOO" [27]. The key screen features considered are the following:

- **Cost:** LCD screens provide good enough resolution and brightness for the purpose of the project and half as expensive as LED screens of the same dimensions without any touch screen feature
- **Modularity:** This LCD screen comes soldered to an Arduino shield that just pops onto and Arduino UNO board without needing any wiring. This allows for an easy and effortless change of component if the screen ever breaks.
- **Touch Screen:** When selecting the screen, it was though that it would be interesting to have some buttons on the screen so that the divers could send commands to the microcontroller in case Bluetooth communication fails.
- **Easy coding:** This screen come with its own set of libraries that allow the user to easily implement GUIs, graphics or images.

The second hardware component needed for the screen to work is the microcontroller to operate it. As there were many pins of the Arduino MEGA board that were in use for the other devices of the Glider, it was decided to use a second Arduino UNO board [28] (**Fig. 64**). This second board takes care of the LCD screen both in current demand and graphic processing power. Then, the two microcontrollers are connected via I2C so that the RTD collected by the sensors on the Arduino MEGA could be display on the LCD screen on the Arduino UNO.

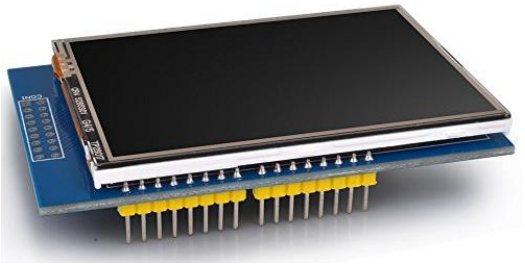


Fig. 63: LCD Screen Arduino Shield

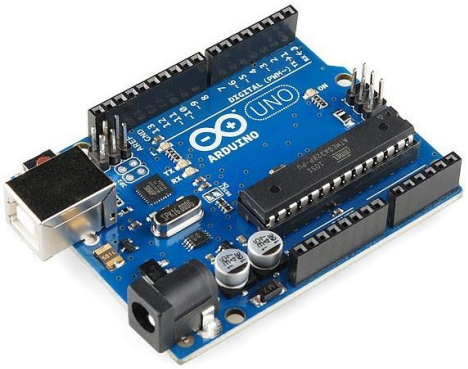


Fig. 64: Arduino UNO Board

5.3.- GUI Design Code

With the screen and microcontroller to be used selected, the Graphical Interface is coded onto the Arduino UNO board. As the screen is mounted on a shield, there is no need for any wiring. Although the mapping of the pins has already been performed by the manufacturer, a small physical modification had to be made in order to connect the 'RESET' pin of the screen to the 'RESET' pin on the Arduino board. This way, Arduino pins 'A4' and 'A5' are free to use for I2C communication. The rest of the wiring stays as default and it is included in the Adafruit's screen libraries [29][30] used for the project (**Fig. 65**).

```
#include <TouchScreen.h>
#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_TFTLCD.h> // Hardware-specific library

// The control pins for the LCD can be assigned to any digital or
// analog pins...but we'll use the analog pins as this allows us to
// double up the pins with the touch screen (see the TFT paint example).
#define LCD_CS A3 // Chip Select goes to Analog 3
#define LCD_CD A2 // Command/Data goes to Analog 2
#define LCD_WR A1 // LCD Write goes to Analog 1
#define LCD_RD A0 // LCD Read goes to Analog 0
#define LCD_RESET A6 // A4 is used for I2C Comm so A6 is defined not to
create interferences. Reset Pin is connected to Arduino's RST pin.

// Pins for the LCD Shield
#define YP A3 // must be analog
#define XM A2 // must be analog
#define YM 9  // digital or analog pin
#define XP 8  // digital or analog pin
```

Fig. 65: LCD Touch Screen libraries and Pin Declaration

With the Screen Shield – Arduino interface pins well defined, some basic parameters for the touch screen and the LCD screen are defined. These also include the renaming of some variables that will be used often in the code to make it easier such as colour names. Then, the LCD screen and Touch Screen are declared in the code as different objects and initialized (**Fig. 66**).

```
#define MINPRESSURE 1
#define MAXPRESSURE 600

// Calibration mins and max for raw data when touching edges of screen
#define TS_MINX 194
#define TS_MINY 130
#define TS_MAXX 909
#define TS_MAXY 905

// Assign names to some common 16-bit color values:
#define BLACK   0x0000
#define BLUE    0x001F
#define RED     0xF800
#define GREEN   0x07E0
#define CYAN    0x07FF
#define MAGENTA 0xF81F
#define YELLOW  0xFFE0
#define WHITE   0xFFFF

Adafruit_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);

TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300);

void TFTsetup(){
  tft.reset();
  uint16_t identifier = tft.readID();
  tft.begin(identifier); //Start communication with TFT screen
}
```

Fig. 66: *Screen Parameter Definition and Set Up*

This first part of the code resets the screen every time it is powered up and re-establish communication between the microcontroller and the screen which are now ready to exchange data. The next part of the code deals with the graphical design of the GUI. To do so, the Adafruit's libraries provide the user with several "drawing" functions. The ones that are mainly used in this project are:

- tft.fillScreen (WHITE)
- tft.drawRect (0, 0, 191, 331, BLACK)
- tft.fillRect (0, 0, 190, 330, BLUE)
- tft.fillRect (0, 0, 20, 330, YELLOW)
- tft.fillTriangle (20, 0, 100, 0, 20, 110, CYAN)
- tft.fillTriangle (20, 0, 97, 0, 20, 107, BLUE)
- tft.fillTriangle (20, 110, 20, 220 , 100, 220, CYAN)
- tft.fillTriangle (20, 113, 20, 220, 97, 220, BLUE)

In every function, each set of two numbers is the exact location, in pixels, of one of the object's vertex. As such, the rectangles are defined by the coordinates of two opposed vertexes whereas in the case of a triangle, the location of the three vertexes is needed. The last argument of the function is the colour of the object to be drawn. Taking this into account and using a trial and error system, the welcome screen for the Glider is coded (**Fig. 67**) and displayed (**Fig. 68**).

```
void WelcomeScreen(){
//Welcome screen design
tft.fillScreen(WHITE);
tft.drawRect(0, 0, 191, 331, BLACK);
tft.fillRect(0, 0, 190, 330, BLUE);
tft.fillRect(0, 0, 20, 330, YELLOW);
tft.fillTriangle(20, 0, 100, 0, 20, 110,
CYAN);
tft.fillTriangle(20, 0, 97, 0, 20, 107,
BLUE);
tft.fillTriangle(20, 110, 20, 220 , 100, 220,
CYAN);
tft.fillTriangle(20, 113, 20, 220, 97, 220,
BLUE);
tft.fillTriangle(20, 220, 100, 220 , 20, 322,
CYAN);
tft.fillTriangle(20, 220, 97, 220, 20, 320,
BLUE);
tft.drawRect(0, 0, 21, 331, BLACK);
tft.setRotation(1);
tft.setCursor(10,85);
tft.setTextColor(BLACK);
tft.setTextSize(3);
tft.println("ALBA GLIDER 3 GUI");
tft.setCursor(70, 150);
tft.setTextSize(2);
tft.setTextColor(YELLOW);
tft.println("Welcome!");
}
```

Fig. 67: *Glider's Welcome Screen Code*

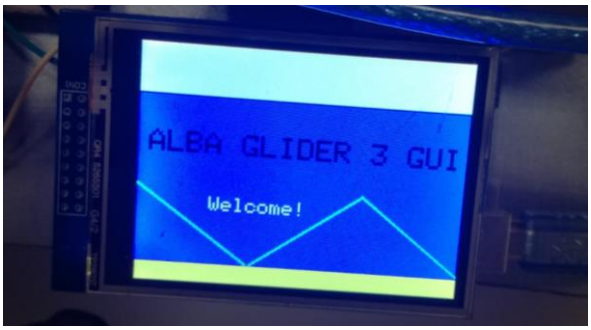


Fig. 68: Glider's Welcome Screen on Display

Apart from drawing shapes and images, the library also allows the user to write text with the following functions:

- tft.setCursor(10, 85)
- tft.setTextColor(BLACK)
- tft.setTextSize(3)
- tft.println("ALBA GLIDER 3 GUI")

This is the common sequence in which any text or number is written on the screen. First, the cursor is placed at one exact location, in pixels, inside the screen. Then, the colour and size of the text is selected. Finally, the text is printed in the selected location with the selected parameters.

These functions help in the design of the next screen (**Fig. 69**), which appears shortly after the welcome screen and it displays the Real Time Data of the Glider. This screen displays the Glider's Pitch Angle, Depth, Temperature, Target Angle and Margin using numeric values coming from the sensors. Besides, the screen also displays the attitude control mass position by means of small graphic representing a bar that increases or decreases depending on the mass' position (**Fig. 70**).

```
void startGUI(){
  tft.fillScreen(BLACK);
  tft.setRotation(1);
  tft.setTextSize(2);
  tft.setTextColor(YELLOW);
  tft.setCursor(0,0);
  tft.println("Vehicle Status");

  //Information write
  tft.setTextSize(2);
  tft.setTextColor(GREEN);
  tft.setCursor(0,20);
  tft.println("Pitch A (Degrees): ");
  tft.setCursor(0,50);
  tft.println("Depth (m): ");
  tft.setCursor(0,80);
  tft.println("Temp (Celsius): ");
  tft.setCursor(0,110);
  tft.println("Set Angle (Degrees): ");
  tft.setCursor(0,140);
  tft.println("Margin (Degrees): ");
  tft.drawRect(40, 170, 240, 30, WHITE);
  tft.setCursor(38,205);
  tft.println("0");
  tft.setCursor(270,205);
  tft.println("MAX");
  tft.setCursor(125,210);
  tft.println("Mass pos");
}
```

Fig. 69: RTD Glider Screen Design Code

Fig. 70: RTD Glider Screen Display Working

On the first version of this screen, using the touch screen feature was also considered and a different interface, including buttons that could send commands to the Glider, was designed. However, due to the inherent difficulty of using a touch screen underwater and the fact that adding a compartment to the AUV's shell would compromise the structure of the vehicle, the idea was discarded and thus not included in this work. The next part will deal with the updating and refreshing of the RTD together with the communication variables used to exchange information between the boards.

5.4.- I2C Channel Configuration and Variables

5.4.1.- I2C Communication Between Arduino Boards

As opposed to most I2C Arduino modules that have their own libraries to deal with the communication procedure, when connecting two Arduino boards, their addresses, roles and data variables to be exchanged in the I2C bus must be declared. This declaration has to be performed on both Arduino boards and all of the variable must have the exact same name for the I2C bus to work. Apart from this, two generic communication libraries [31][32] are used for the connection to work (**Fig. 71-72**).

```
#include <SoftwareSerial.h>
#include <EasyTransferI2C.h>

EasyTransferI2C UNO;

#define MASTER_ADDR 7  //I2C Master
#define UNO_ADDRESS 8 //I2C slave address

struct SEND_DATA{
    int depth;
    int temp;
    double inAngle;
    int posRail;
    double setAngle;
    int margin;
    int hours;
    int minutes;
    int seconds;
};

SEND_DATA science;

void startControl()
{
  Wire.begin(MASTER_ADDR);
  UNO.begin(details(science), &Wire);
```

Fig. 71: I2C Bus Configuration on Arduino MEGA (Master)

```
#include <Wire.h>
#include <EasyTransferI2C.h>

EasyTransferI2C UNO; //Name of slave device

struct RECEIVE_DATA{
    int depth;
    int temp;
    double inAngle;
    int posRail;
    double setAngle;
    int margin;
    int hours;
    int minutes;
    int seconds;
};

RECEIVE_DATA science;

#define UNO_ADDRESS 8

void setup() {

  Serial.begin(9600);
 //I2C Comm
  Wire.begin(UNO_ADDRESS); // join i2c bus
  with address #8
  UNO.begin(details(science), &Wire);
```

Fig. 72: I2C Bus Configuration on Arduino UNO (Screen)

With the help of these libraries, the procedure to communicate both boards is the following:

1. Define the name of the "Slave" device on both bords *('UNO' in this case*).
2. Define the Bus Address for both the "Master" and the "Slave" (*7 for MEGA, 8 for UNO board*).
3. Create an identical data structure that contains exactly the same variables in each board. The name of the structure may differ from one board to another (*RECEIVE/SEND_DATA*).
4. Define the name that the structure will have on the bus. This name must be exactly the same for both boards (*science*).
5. On the set up of each board, enter the bus with the assigned address *("Wire.begin (…)")* and start the communication with the declared "Slave" using the declared data structure on both boards *("UNO.begin(details(science), &Wire)")*.

5.4.2.- Screen Real Time Data Update

Once the communication is stablished, the screen module will be constantly checking for any information update on the I2C bus. To do so, a receive event is created so that each time the Arduino UNO detects new data on the bus, the screen variables are updated (**Fig. 73**).

```
void setup() {

  Wire.onReceive(receiveEvent); // register event
}

void loop() {
  if(UNO.receiveData()){
    d = science.depth;
    t = science.temp;
    a = science.inAngle;
    pos = science.posRail;
    setA = science.setAngle;
    mar = science.margin;
    //h = science.hours;
    //m = science.minutes;
    //s = science.seconds;
  }

// function that executes whenever data is received from master

void receiveEvent(int numBytes) {
  //UpdateDate(h, m, s);
  UpdateData(a, d, t, pos, setA, mar);
  }
```

***Fig. 73**: Screen Module Update Routine Code*

Following the spirit of the rest of the project, the Arduino UNO code for the screen is also programmed in a modular way. Functions as "UpdateData" or "UpdateDate" are user created and they take care of refreshing the values passed to the function on the screen (**Fig. 74**). As said before, the program checks for any data received through the bus every cycle and updates the variables if so. The values coming from the RTC (*h, m and s*) were commented out of the code due to the fact that it made the communication very slow causing a great loss of RTD.

```
void UpdateData (double angle, int dep, int temp, int Rpos, double sAngle, int margin){
  tft.setTextSize(2);
  tft.setTextColor(WHITE, BLACK); //Print with black background for refresing

  //Slider representation in GUI
  tft.fillRect(41, 171, 238, 28, BLACK);
  rail=map(Rpos, 0, 120, 1, 238);
  tft.fillRect(41, 171, rail, 28, RED);

  tft.setCursor(250,20);
  tft.println(angle);
  tft.fillRect(249, 49, 40, 29, BLACK);
  tft.setCursor(250,50);
  tft.println(dep);
  tft.setCursor(250,80);
  tft.println(temp);
  tft.setCursor(250,110);
  tft.println(sAngle);
  tft.setCursor(250,140);
  tft.println(margin);
}
```

***Fig. 74**: Screen Module Data Update Function*

The "UpdateData" function uses the same drawing capabilities explained on part 4.3. However, to fill the rectangle for the mass position (**Fig. 70**), one of the coordinate values for the "tft.fillRect" function is dynamic and depends on the I2C variable called "Rpos". The rest of the RTD values are simply printed on their assigned places. In order to ensure a proper refresh of the screen, a black rectangle covering the values is drawn before printing the new ones.

The final aspect of the communication between the boards is to know how and when the data is sent from the sensor board (Arduino MEGA) to the screen module (Arduino UNO). Inside the Arduino MEGA code, the sensor values are stored in local variables. Then, the I2C communication variables are updated with the local values in the following cases: when the program starts, every time the motor moves or every 20 program cycles if none of these happen. Once updated and ready to be transmitted, they are sent though the bus to the screen using the function 'DEVICENAME.sendData("DEVICEADDRESS")' (**Fig. 75**).

```
void testControl()
{

    science.depth = depth;
    science.temp = tempA;
    science.inAngle = inAngle;
    science.posRail = posRail;
    UNO.sendData(UNO_ADDRESS);

}
```

***Fig. 75:** Arduino MEGA I2C Data Update and Send Code*

6.- CTD Sensor Module

6.1.- Introduction

The CTD module is a cluster of sensors that allow for the measurement of 3 main magnitudes: Conductivity, Temperature and Depth [33] (**Fig. 76**[9]). This last magnitude is obtained from water pressure values and a simple transfer function. The measurement of these magnitudes is the heart of the Glider as the data gathered by this set of sensors will allow for the monitoring of the area that the vehicle is exploring and also provide a ground on which environment predictions can be based. Inside the Glider, this module will be safely located at the nose of the vehicle (**Fig. 77**) and close to the water that surrounds it in order to obtain the most accurate measurements possible and the water samples for the conductivity measurements.

Given the importance of this data, it is mandatory to store it, as frequently as possible, in a safe place during the Glider's journey. For this reason, the Arduino UNO board that collects the data and controls the sensors inside the CTD module will be connected via I2C to the Arduino MEGA board of the Glider so that the sensor data can be stored inside an SD Card for later analysis.

As the development and coding of this module was performed by another colleague student called José Luis Pérez, this part will focus on the connection of the two boards and the exchange of data between them. Data storage and database structure will be further assessed in Part 6 of this work.

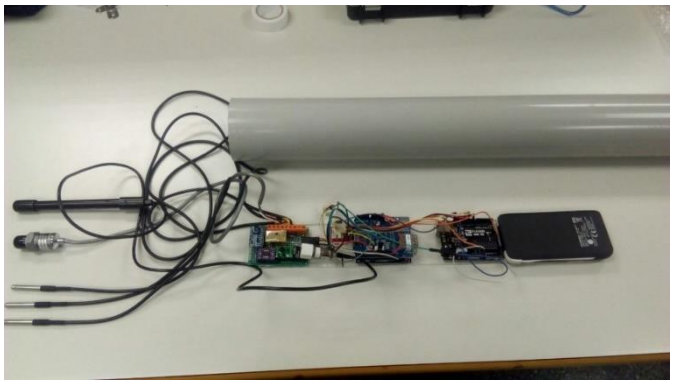


Fig. 76: *CTD Module Controller and Sensors Setup*

Fig. 77: *Nose Hull for CTD and Optical Comm System Housing*

6.2.- I2C Communication Procedure

As it has been explained in part 3.5 of this document, the Arduino boards are connected together using the I2C interface and their corresponding SCL and SDA pins (**Fig. 61**). The coding needed to establish the communication is almost identical to the one used for the display screen and the Arduino MEGA board, explained in part 4.4.1. This means that the libraries and data structure used are the same. However, the variables to be transmitted and the device address for the CTD module change (**Fig. 78**).

[9] Image provided by Jose Luis Pérez from his work.

```ino
#include <Wire.h>
#include <EasyTransferI2C.h>

EasyTransferI2C CTD;
#define MASTER_ADDR 10  //I2C Master
#define CTD_ADDRESS 9  //I2C CTD sensors slave address

struct CTD_DATA{
  int16_t depth;
  float tempA;
  float tempB;
  float tempC;
  float conduct;
};

CTD_DATA ctd_data;

//CTD sensor variable for reception
int16_t depth=0;
float tempA=0, tempB=0, tempC=0, conduct=0;

unsigned long MesMillis = 0;

void receiveEvent(int numBytes){
}

void setup() {
  Serial.begin(9600);
  Wire.begin(MASTER_ADDR);
  CTD.begin(details(ctd_data), &Wire);
  Wire.onReceive(receiveEvent);
}
```

```ino
// For communicating with another Arduino by I2C
#include <Wire.h>
#include <EasyTransferI2C.h>
#define CTD_ADDRESS 9   //I2C slave address
#define MASTER_ADDR 10  // I2C master address

//I2C communication for CTD
EasyTransferI2C CTD;

struct CTD_DATA{
    int16_t depth;
    float tempA;
    float tempB;
    float tempC;
    float conduct;
};

CTD_DATA ctd_data;

void setup(void) {

  // Communication
  Wire.begin(CTD_ADDRESS);
  Wire.onReceive(receive);
  CTD.begin(details(ctd_data), &Wire);
}
```

Fig. 78: *I2C Bus Configuration for Arduino MEGA (left) and CTD Module (right)*

Opposed to the Arduino MEGA – Display Screen connection, the master device is now receiving data from the CTD module instead of sending local data to the screen module. Thus, the receive event is inside the Arduino MEGA code shown. The data structure that will be used is called “ctd_data” on both boards and it includes the 5 measurements coming from the CTD module sensors. In order to obtain the most faithful reading from the module, three different thermometers are used to obtain the average temperature of the water. These three measurements are related to variables ‘tempA’, ‘tempB’, ‘tempC’.

Every cycle, the program checks for new data available on the I2C bus and, if there is new data, the local variables inside the Arduino MEGA board are updated and stored in the SD Card (**Fig. 79**).

```ino
void loop() {
  // put your main code here, to run repeatedly:
  if(CTD.receiveData()){
    depth = ctd_data.depth;
    tempA = ctd_data.tempA;
    tempB = ctd_data.tempB;
    tempC = ctd_data.tempC;
    conduct = ctd_data.conduct;
    MesMillis = millis();
    WriteSensorValues(MesMillis, depth, tempA, tempB, tempC, conduct);
  }
}
```

```ino
void loop(void) {

  updateCTD();
  sendDataCTD();
  delay(100);  // Wait 100 milliseconds.
}

void sendDataCTD()
{
    ctd_data.depth    = getPressure();
    ctd_data.tempA    = getTempA();
    ctd_data.tempB    = getTempB();
    ctd_data.tempC    = getTempC();
    ctd_data.conduct = getConductivity();

    CTD.sendData(MASTER_ADDR);
}
```

Fig. 79*: Arduino MEGA Data Reception (left) and CTD Variables Update & Sending Code (right)*

6.3.- CTD Data Log

In order to keep all the information coming from the CTD sensors well organized, each time the Arduino MEGA is powered up, a file called 'LOGCTD.csv' is created, which will store the readings coming from the sensors using comma separated values. This file is updated every time the local values on the Arduino MEGA are updated (**Fig. 79**) using the 'WriteSensorValues' function. This user defined function will be further explained in the next part of this work.

The update of the CTD sensor values is performed every 100 milliseconds in order to obtain a solid real time data base. With this many data logs, time-variant graphs can be obtained from the sensors measurements in order to analyse the data and perform predictions based on it.

6.4.- Honeywell Depth Sensor

As the focus of this part is made on sensor data, it is important to mention that another standalone pressure sensor is included in the vehicle. In the same way as in the CTD pressure sensor, this sensor transforms an atmospheric pressure reading into a depth value in meters using the formulas found on the datasheet of the Honeywell ASDX Sensor Series [34] (**Fig. 80**). This depth value is then used for display on the mobile APP and logged into a ".csv" file for later analysis.

```
#include "sensors.h"
#include <Wire.h>
#include "DHT.h"
#define I2C_PRESSION 40 //Unique bus address for
pressure sensor
#define MEGA
#define EXT_REF 0
#define ADC_HONEYWELL 2 // Arduino analog input pin

//Pressure sensor
byte msb, lsb = 0;
int press = 0;
int out_Max = 14745;
int out_Min = 1600;
int P_max = 90; //psi (max.: 6.12 atm)
int P_min = 15; //psi (max.: 1.02 atm)
int P_out = 0;
```

```
void getdata(byte *a, byte *b)
{
  Wire.requestFrom(I2C_PRESSION,2);
  while (Wire.available()){
  *a = Wire.read();          //first byte recieved
stored here
  *b = Wire.read();          //second byte recieved
stored here
  }
}

float getPressure()
{
  getdata(&msb,&lsb);
  press = msb;
  press = (press << 8) + lsb;
  P_out = (((press - out_Min)*(P_max -
          P_min))/(out_Max - out_Min)) + P_min;
  //Conversion found in datasheet
  return P_out;
}
```

Fig. 80: *Honeywell Pressure Sensor Data Adquisition and Handling Code*

As with the CTD module, this sensor is also connected to the Arduino by the I2C bus which sends over the analog readings to be interpreted. However, the data acquisition code is included as a standalone module which means that the variable containing the depth value must be exported to the main control code. This is achieved by using the same structure procedure as for the I2C communication between boards (**Fig. 78**) only this time the "struct" only contains one variable. Then, on the main code, the data acquisition function is called at the same time the display screen is updated and so are the variable values. Then, they are logged into the "LogHONEY.csv" database which we be further explained in the next part.

7.- SD Card Data Log

7.1.- Introduction

In this part, the date storage capabilities of the Glider will be assessed together with explanations on the libraries and code used to perform the data log into the SD Card. It is important to remember that the main idea behind this project is to a create a low-cost AUV which collects data from its environment to be used for biological experimentation, natural disaster prediction or even water habitat damage assessment after said disaster.

For this reason, every subsystem on the Glider has been explained before in order to know where every piece of data is coming from and just focus on the design of the databases and the data analysis afterwards. Thus, this part will be organized as follows:

- Data storage libraries and code
- Database files structure
- Data analysis

7.2.- Data Storage Libraries and Code

As explained in part 3.4, the SD card adapter module uses the "SD.h" [23] library in order to manage the SPI communication with the Arduino MEGA board and it also provides some basic data storage and file management functions. However, these basic functions must be combined in a way that the collected data can be properly displayed for analysis. For this reason, several new functions are created (**Fig. 81**) in order to structure the database in a suitable fashion. These functions are declared as a standalone module which can be later introduced into the main code.

```
#include <SD.h>
#include <Arduino.h>

void SDSetup();
void CreateMotorLogFile();
void CreateSensorLogFile();
void CreateHoneyWellLogFile();
void WriteMotorValues (int RTC_mins, int RTC_sec, unsigned long MesMillis, int motor_FW, int motor_BW);
void WriteSensorValues (int RTC_mins, int RTC_sec, unsigned long MesMillis, int16_t depth, float tempA, float tempB, float tempC, float conduc);
void WriteHoneyValues (int RTC_mins, int RTC_sec, unsigned long MesMillis, int depth);
```

Fig. 81: *SD Card Module User Defined Functions*

The new user defined functions are divided in 3 main groups:

- **SD card setup:** this function checks that the SD Card module is properly connected to the Arduino MEGA board using the "SD.begin" library function and the chip select (CS) pin defined earlier in the code. If the function returns a 0, the functions creates a card failure warning message for the user. Otherwise, the function sets the internal variable "SDin" to 1, which allows for the execution of the rest of the code, and tells the user that the card is ready (**Fig. 82**).

- **File generation:** this function is in charge of creating a new data log file with the specified name using the function "SD.open". Once created, it prints the header row containing the name of each column in the new file. Then, the file is closed and ready to be filled with data. In the event that the file cannot be created, a warning message is shown (**Fig. 83**).
- **Data write:** whenever this function is called, it prints a new line in the specified file with the values introduced as arguments. The functions requires an argument for each column or field in the file (**Fig. 84**).

This section will focus on demonstrating the basic operating principles of each of the functions and thus only one example of each will be included.

```
void SDSetup(){

 Serial.println("Initializing Card");
  pinMode(CS_pin, OUTPUT);

  //Check if card is ready
  if(!SD.begin(CS_pin)){
    Serial.println("Card Failed!!");
    SDin=0;
    return;
  }
  Serial.println("Card Ready");
  SDin=1;
}
```

Fig. 82: *SD Card Setup Function Code*

```
void CreateMotorLogFile(){
  File logFile = SD.open("LogSTEPL.csv", FILE_WRITE); //Movement of the longitudinal tray
  if (logFile){
    logFile.println(", , , ,"); //Blank line
    String header = "RTC_Min,   RTC_sec,   Millis,   Motor_FW,   Motor_BW";
    logFile.println(header);
    logFile.close();
  }else if (SDin==1){
    Serial.println("Couldn't open log file");
  }
}
```

Fig. 83: *File Creation Function Code*

```
void WriteMotorValues (int RTC_mins, int RTC_sec, unsigned long MesMillis, int motor_FW, int motor_BW){
  //CSV format data string
  String dataString = String(RTC_mins)+ ",    " + String(RTC_sec) + ",    " + String(MesMillis) + ",    " + String(motor_FW) + ",    " + String(motor_BW);

  //Open file to write to, only one file open at a time
  File logFile = SD.open("LogSTEPL.csv", FILE_WRITE);
  if(logFile){
    logFile.println(dataString);
    logFile.close();
  }else if (SDin==1){
    Serial.println("Couldn't access file");
  }
}
```

Fig. 84: *Data Write Function Code*

7.3.- Database File Structure

The basic structure for every database is the same: values are introduced separated by a comma each and the header row includes a short description of each of the fields. Being real time databases, the first row of time values is logged right at the start of the program execution, during the setup. The following sections will provide more insight details about each database and its data gathering goals.

7.3.1.- Stepper Motor Log File

In order to monitor the power that the mass trimming motors consume, a log file database is created ("LogSTEPL.csv"). This database records the amount of time that the motor is moving while taking into account its direction of movement. To do so, it has 3 fields containing real time values up to the millisecond subdivision (RTC minutes, RTC seconds and Milliseconds) and 2 fields (Motor Forwards and Motor Backwards) that indicate whether the motor is moving, in one direction or another, or not (**Fig. 83**).

Motor FW	Motor BW	Pitch Angle Error	Meaning
0	0	0	Motor is stopped
0	1	POSITIVE	Motor moving backwards
1	0	NEGATIVE	Motor moving forwards
1	1	-	No physical meaning

Fig 85: *Motor Database Significance Table*

Knowing how to interpret the database, it is important to know when the file is written within the main code. As the focus is made on the motor operation, the file is written every microcontroller cycle whether the motor moves or not. The values for the motor direction columns are updated accordingly (**Fig. 85**)

Within this database, it would also be interesting to include two additional fields to keep track of the target angle value and the vehicle's pitch angle. This setup would show that the motor stops when the target angle is reached. This addition would also show the proper function of the Proportional Controller that modulates the rotation speed of the motor.

7.3.2.- CTD Sensors Log File

As it has been explained in part 5 of this document, data collection is the upmost duty of this Glider. Regardless of the origin of the data, whether it comes from the CTD module integrated in the vehicle or any external source, it needs to be interpreted properly in order to make good predictions or situation assessments. That is why an independent log database ("LogCTD.csv") is created. This database (**Fig. 86**) contains 3 fields that record the real time when the file is written (RTC_Min, RTC_Sec and Millis) and 5 other fields corresponding to each of the CTD modules sensors (Depth, Temperature A, Temperature B, Temperature C and Conductivity).

```
void CreateSensorLogFile(){
  File logFile = SD.open("LogCTD.csv", FILE_WRITE); //Movement of the longitudinal tray
  if (logFile){
    logFile.println(", , , , , , ,"); //Blank line
    String header = "RTC_Min,   RTC_sec,   Millis,   Depth,   TemperatureA,   TemperatureB,   TemperatureC,   Conductivity";
    logFile.println(header);
    logFile.close();
  }else if (SDin==1){
    Serial.println("Couldn't open log file");
  }
}
```

Fig. 86: *CTD Sensor Database Structure Definition and Creation*

This set of data will allow for the plotting of the CTD sensor values using a real time scale. To ensure that the data records are as close as possible to a real time monitoring of the reality, the CTD module sends data to the Arduino MEGA board, which records the instant value of the sensors, every 100 milliseconds.

7.3.3.- Honeywell Depth Sensor Log File

Previously explained in part 5.3, the Honeywall depth sensor provides auxiliary pressure readings, useful for the navigation of the Glider. As this sensor is not part of the CTD module, a new database ("LogHONEY.csv") containing 3 fields for real time records (RTC_min, RTC_sec and Millis) and 1 field for the Glider's depth values overtime (Depth), is created (**Fig. 87**).

```
void CreateHoneyWellLogFile(){
  File logFile = SD.open("LogHONEY.csv", FILE_WRITE);
  if (logFile){
    logFile.println(", , , "); //Blank line
    String header = "RTC_Min,   RTC_sec,   Millis,   Depth";
    logFile.println(header);
    logFile.close();
  }else if (SDin==1){
    Serial.println("Couldn't open log file");
  }
}
```

Fig. 87*: Honeywell Sensor Database Structure Definition and Creation*

The log frequency is the same as the update frequency of the values on the display screen explained in part 4 of this document. As such, the log file is written every 20 microcontroller cycles with the depth value ("depthH") returned from the function named "cycleScience" (**Fig. 88**), which ultimately uses the "getPressure" function seen earlier (**Fig. 80**), to obtain the right reading.

```
void testControl()
{
  struct sciData theSci;
  theSci = cycleScience(inAngle, tempA);
  //Honeywell dataLOG
  MesMillis = millis();
  WriteHoneyValues(m, s, MesMillis, theSci.depthH);  //Log into SD card

  science.depth = depth;
  science.temp = tempA;
  science.inAngle = inAngle;
  science.posRail = posRail;
  UNO.sendData(UNO_ADDRESS);
}
```

Fig. 88: *SD Card Honeywell Sensor Log Function*

7.4.- Data Analysis

As the databases are quite similar except for the number of fields and the kind of data they contain, the arrangement of the data for its analysis is the same. For this reason, this section will only focus on one of the files.

The program chosen for the data analysis is Excel because it is powerful enough for the kind of graphics it is intended to generate and easy to use. Then, the first step is to arrange the CSV values into different columns (**Fig. 89**).

RTC_Min	RTC_sec	Millis	Motor_FW	Motor_BW
30	43	21950	0	0
30	43	22120	0	0
30	43	22227	1	0
30	43	22734	0	0
30	43	22772	0	0
30	43	22881	1	0
30	43	23388	0	0
30	43	23425	0	0
30	43	23533	1	0
30	43	24041	0	0
30	43	24078	0	0
30	43	24186	1	0
30	43	24693	0	0
30	43	24731	0	0
30	43	24843	1	0
30	43	25350	0	0
30	43	25388	0	0

***Fig. 89**: Database Records displayed in Columns in Excel*

Once the real time values are manipulated in a suitable way for the data analysis by, for example, joining the minutes and seconds to create the labels for the graph, each of the data fields is introduced in the graph as an independent series. This leads to a graphic representation of a big number of records (**Fig. 90**).

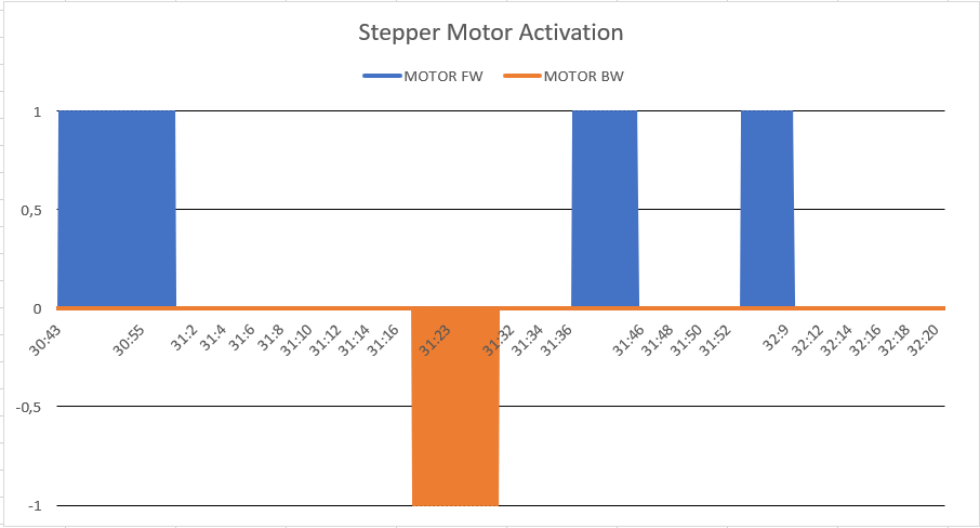


***Fig. 90:** Motor Activation Excel Graph*

8.- Conclusion & Results

8.1.- Results

Due to several time and budget constraints, the control system implemented in this project has not been tested underwater. However, every system designed has proved to be working in the lab environment. To perform a detailed results analysis of each module, they will be assessed independently:

1. **Arduino MEGA Board:** as the 'brain' of the systems designed in this project, the Arduino MEGA has proved to be powerful enough to properly handle every subsystem according to the algorithm written. However, computing limitations were found when attempting to display the data coming from the RTC module on the screen. Besides, it is important to mention that the function in charge of moving the stepper motors blocks the code execution until the movement is completed. These two limitations should be considered when analysing the RTD collected by the Glider.
2. **Stepper Motor Control:** the hardware items selected and designed (PCB) work flawlessly together proving to have enough torque and structural strength to pull the trimming mass, even at a 90° angle. The control and communication algorithms designed works as expected, allowing for the automatic and manual operation of the motors.
3. **BT Module & Communications:** despite its limited range, the sending of BT commands and the reception of RTD on the mobile device works properly together with the designed APP. The rest of the communication procedures fulfil their jobs as expected allowing for the communication of every module with the Arduino MEGA.
4. **Display Screen:** throughout the design process, several options have been studied for the display screen. Before being used as an assistance RTD display device for the Glider's operator, the screen also allowed for command sending, taking advantage of its touchscreen feature. However, the screen ended up being too full of different buttons and so the control functionalities were laid off in benefit of the display functionality. Both the communication and data update on the screen worked properly.
5. **CTD Module:** given the fact that this module is meant to be tested mostly underwater and it was designed by another college, only the communication and update capabilities are assessed. As such, the communication proved to be reliable, allowing for a regular data update.
6. **SD Card Data Log:** despite having the log files written many times in one microcontroller cycle, the module copes properly with the many data accesses. The file creation also proved to work properly creating the 3 data files with the right structure database.

8.2.- Conclusions

- Stepper motors provide a compact and viable solution to the trimming system of the Glider while ensuring higher accuracy and torque values despite needing more complex controllers.
- PCB design can be very useful when no manufacturer is able to offer the solution needed. The use of PCB specific programs makes the design and fabrication process easier and accessible to a great number of users. Together with the low fabrication cost offered by some websites, custom PCBs make the design cleaner and more reliable.
- Wireless communications are key as the Glider will be navigating without any human input. Thus, the RTD update transmission has to be reliable and secure in order to keep the vehicle located and monitored at all times.
- Modular coding helps organize the subsystems designed and makes the code more readable and understandable whether you are familiar with it or not. Besides, coding each module independently allow for the testing of these systems without interfering with the main code. In a project with such amount of coding, modules are vital.
- Within the laboratory environment, it has been proved that the control and operation of an AUV vehicle is possible using low-cost Arduino modules and microcontrollers without compromising its reliability or functionality.
- Different control, display, communication and storage systems have been designed and tested under the scope of this project thus fulfilling the objectives explained in Part 1.3. Besides, this project has proved to be a good blend of every skill acquired within the Electronic and Automatics Degree such as control theory, coding or digital/analog electronics.

9.- References

[1] Claus B., Bachmayer R., Williams C.D., "Development of an auxiliary propulsion module for an autonomous underwater glider, Proc. of the Institution of Mechanical Engineers", Part M: *Journal of Engineering for the Maritime Environment*, 224 (4) (2010), pp. 255-266

[2] Davis R.E., Eriksen C.C., Jones C.P., "Autonomous Buoyancy- driven Underwater Gliders", *The Technology and Applications of Autonomous Underwater Vehicles*, Taylor and Francis, London (2002): G. Griffiths (Ed.), pp. 37-58

[3] Alvarez A., Caffaz, A. Caiti, G. Casalino, L. Gualdesi, A. Turetta, R. Viviani Folaga, "A low-cost autonomous underwater vehicle combining glider and AUV capabilities", *Ocean Engineering*, 36 (1) (2009), pp. 24-38

[4] Webb D.C., Simonetti P.J., Jones C.P., "SLOCUM, an underwater glider propelled by environmental energy", *IEEE Journal of Oceanic Engineering*, 26 (2001), pp. 447-452

[5] Glenn S., Schofield O., Kohut J., McDonnell J., Ludescher R., Seidel D., Fanjul E., "The Trans-Atlantic Slocum Glider Expeditions: A Catalyst for Undergraduate Participation in Ocean Science and Technology", *Marine Technology Society Journal*, 45 (1) (2011), pp. 52-67

[6] *Ibidem.*

[7] Busquets J., Busquets D.., Busquets J.V., "Combined Gas-Fluid Buoyancy System for Improved Attitude and Maneuverability Control for Application in Underwater Gliders", *IFAC-PapersOnLine*, 48-2 (2015) 281–287

[8] Anonymous, "Adafruit Motor Shield", *Adafruit Explore & Learn,* (2012), https://learn.adafruit.com/adafruit-motor-shield/overview , accessed January, 2018

[9] Anonymous, "DRV8825 Stepper Motor Driver Carrier, High Current, Item #2133", *Pololu Stepper Motor Drivers*, https://www.pololu.com/product/2133, accessed January, 2018

[10] https://circuitmaker.com accessed February, 2018

[11] https://www.altium.com/es/ accessed February, 2018

[12] https://www.seeedstudio.com/fusion_pcb.html accessed February, 2018

[13] Oscar Liang, "HOW TO USE GY80 ARDUINO – ADXL345 ACCELEROMETER", (2014), https://oscarliang.com/use-gy80-arduino-adxl345-accelerometer/ , accessed January, 2018

[14] Anonymous, "Understanding Kalman Filters", *Video and Webminar Series*, (2017) https://www.mathworks.com/videos/series/understanding-kalman-filters.html, accessed January, 2018

[15] T. Lacey, "Tutorial: The Kalman Filter", *Computer Vision*, http://www.cc.gatech.edu/classes/cs7322-98-spring/PS/kf1.pdf , accessed November, 2018

[16] Cebrián Abellán, A., "Sistema de desplazamiento de masas para el control de orientación de un Glider submarino", *DISCA*, (2018)

[17] Laurentiu Badea, "Arduino library for A4988, DRV8825, DRV8834, DRV8880 and generic two-pin (DIR/STEP) stepper motor drivers", *Stepper Driver*, https://github.com/laurb9/StepperDriver, accessed February, 2018

[18] Luis Llamas, "CONECTAR ARDUINO POR BLUETOOTH CON LOS MÓDULOS HC-05 Ó HC-06", *Tutoriales Arduino*, (2015), https://www.luisllamas.es/conectar-arduino-por-bluetooth-con-los-modulos-hc-05-o-hc-06/, accessed March, 2018

[19] Shah Saifur Rahman, "AT Command Mode of HC-05 and HC-06 Bluetooth Module", (2017), https://www.instructables.com/id/AT-command-mode-of-HC-05-Bluetooth-module/, accessed March, 2018

[20] MIT, "APP INVENTOR 2", http://appinventor.mit.edu/explore/, accessed November, 2018

[21] Anonymous, "Scratch (programming language)", https://en.wikipedia.org/wiki/Scratch_(programming_language), accessed January, 2019

[22] Mike Grusin, "Serial Peripheral Interface (SPI)", *Tutorials*, https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all, accessed November, 2018

[23] Anonymous, "SD Library", *Reference*, https://www.arduino.cc/en/reference/SD, accessed April 2018

[24] Anonymous, "I^2C", https://en.wikipedia.org/wiki/I%C2%B2C, accessed January, 2019

[25] Anonymous, "I2C", *Aprendiendo Arduino,* https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/, accessed January, 2019

[26] Dmainmon, "UNO R3 2.8 TFT Touch Screen With SD Card Socket for Arduino Board Module", https://www.instructables.com/id/UNO-R3-28-TFT-Touch-Screen-With-SD-Card-Socket-for/, accessed April, 2018

[27] https://www.elegoo.com/, accessed January, 2019

[28] https://store.arduino.cc/arduino-uno-rev3, accessed February, 2018

[29] Phillip Burgess, "Adafruit GFX Graphics Library", *Adafruit Explore & Learn*, https://learn.adafruit.com/adafruit-gfx-graphics-library/overview, accessed April, 2018

[30] Anonymous, "TFTLCD-Library", *Adafruit*, https://github.com/adafruit/TFTLCD-Library, accessed April, 2018

[31] Anonymous, "Wire Library", *Reference*, https://www.arduino.cc/en/Reference/Wire, accessed March, 2018

[32] Bill Porter, "Arduino Easy Transfer", https://github.com/madsci1016/Arduino-EasyTransfer, accessed March, 2018

[33] Anonymous, "CTD (instrument)", https://en.wikipedia.org/wiki/CTD_(instrument), accessed April, 2018

[34] Honeywell, "ASDX Series Silicon Pressure Sensors", https://sensing.honeywell.com/honeywell-sensing-asdx-series-digital-pressure-sensors-product-sheet-008095-13-en.pdf, accessed May, 2018

DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

ESCUELA TÉCNICA SUEPERIOR DE INGENIERÍA DEL DISEÑO

UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEVELOPMENT OF THE CONTROL ELECTRONICS FOR THE NAVIGATION OF AN UNMANNED SUBMARINE WITH ARDUINO

2. REQUIREMENTS

TRABAJO DE FIN DE GRADO:

Grado en Ingeniería Electrónica Industrial y Automática

Autor: Fco. Javier Pérez Villaplana

Director: Jose Vicente Busquets Mataix

Codirector: Javier Busquets Mataix

VALENCIA, ABRIL 2019

Index

1.- Requirements

The requirements that this project must fulfil, within the whole structure of the Glider, are the following:

1. The attitude control system designed must be able to change the orientation of the Glider, both in the pitch and roll axis, according to the user specifications. This needs to be achieved by the displacement of two inertial masses that change the centre of mass of the vehicle and thus produce a shift in its orientation. This system, joined with the variable buoyancy system, allow for the forward movement of the Glider.

2. The Glider must be able to transmit real time information as well as storing this information for later computer analysis.

3. The Glider must have a communication interface for the user to change its internal parameter and set new targets for the pitch and roll angles. This should be achieved by a Bluetooth module and a GUI designed for a mobile APP.

4. Throughout the whole design process, the cost of the different modules, sensors and actuators must be as low as possible, using the already available materials in the lab.

5. In order to ensure an easy repair or replacement of the components in the control system, modularity must be applied to every sensor and actuator connected to the Arduino board. This way, whenever something fails, it will just be a Plug & Play repair.

6. The programming language must be C, with the Arduino syntaxis, so that every piece of code can be added to the whole Glider project after.

7. Modularity must also be applied to the code, separating each system, sensor or actuator into an independent module. This should be achieved with the creation of functions that will be called on the main part of the code.

8. Space economy should also be considered, and hardware component must have the smallest size possible. This can be achieved by including as much hardware as possible within a PCB Arduino shield.

9. Power consumption and build quality must also be assessed in the design and implementation of the hardware as well as in the selection of the power source for the system.

DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

ESCUELA TÉCNICA SUEPERIOR DE INGENIERÍA DEL DISEÑO

UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEVELOPMENT OF THE CONTROL ELECTRONICS FOR THE NAVIGATION OF AN UNMANNED SUBMARINE WITH ARDUINO

3. BUDGET

TRABAJO DE FIN DE GRADO:

Grado en Ingeniería Electrónica Industrial y Automática

Autor: Fco. Javier Pérez Villaplana

Director: Jose Vicente Busquets Mataix

Codirector: Javier Busquets Mataix

VALENCIA, ABRIL 2019

Index

1.- Material Cost

Description	Unit	Cost (€/unit)	Units	Total Cost (€)
Arduino MEGA 2560	u	35	1	35
Arduino UNO	u	20	2	40
NEMA 17 Stepper Motor	u	12	2	24
DRV8825 Controller	u	8	2	16
3 DIP Switch	u	0,85	2	1,7
Electrolytic Capacitor	u	0,35	3	1,05
2 Pin Terminal Block	u	0,5	5	2,5
8 Pin Socket	u	0,25	6	1,5
6 Pin Socket	u	0,25	2	0,5
GY-80 IMU 10 DOF	u	7	1	7
HC-06 BT Module	u	3,8	1	3,8
SD Card Adapter Module	u	2	1	2
RTC Module	u	2,5	1	2,5
2.8'' LCD Touch Screen Shield	u	15	1	15
Honeywell ASDX Pressure Sensor	u	55	1	55
PCB	u	1,5	1	1,5
USB Li-Po Battery	u	18	5	90
			TOTAL	**300**

2.- Development & Labour Cost

Description	Unit	Cost (€/unit)	Units	Total Cost (€)
Project Study & Hardware Selection	h	25	20	500
Software Design	h	25	15	375
PCB & Hardware Design	h	25	40	1000
Coding & Debugging	h	25	120	3000
Assembly	h	25	10	250
Testing	h	25	25	625
Documentation	h	25	130	3250
			TOTAL	9000

3.- Budget

Description	Unit	Cost (€/unit)	Units	Total Cost (€)
Material Cost	u	300	1	300
Development & Labour Cost	u	9000	1	9000
			TOTAL	9300

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

ESCUELA TÉCNICA SUEPERIOR DE INGENIERÍA DEL DISEÑO

UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEVELOPMENT OF THE CONTROL ELECTRONICS FOR THE NAVIGATION OF AN UNMANNED SUBMARINE WITH ARDUINO

4. DIAGRAMS & SCHEMATICS

TRABAJO DE FIN DE GRADO:

Grado en Ingeniería Electrónica Industrial y Automática

Autor: Fco. Javier Pérez Villaplana

Director: Jose Vicente Busquets Mataix

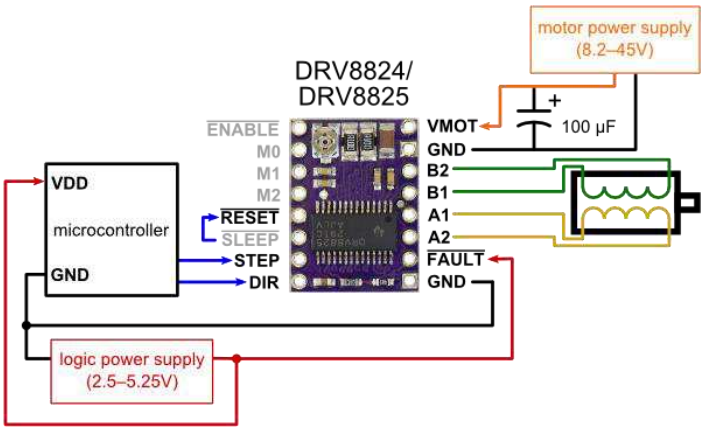
Codirector: Javier Busquets Mataix

VALENCIA, ABRIL 2019

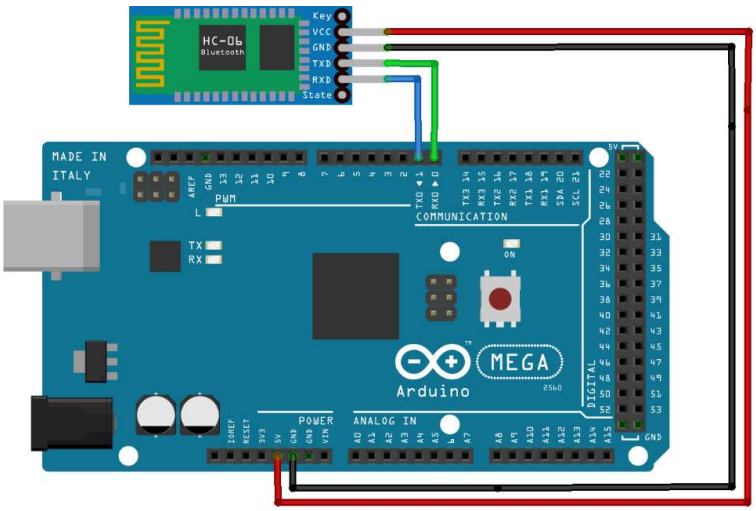
Index

1.- Connection Schematics

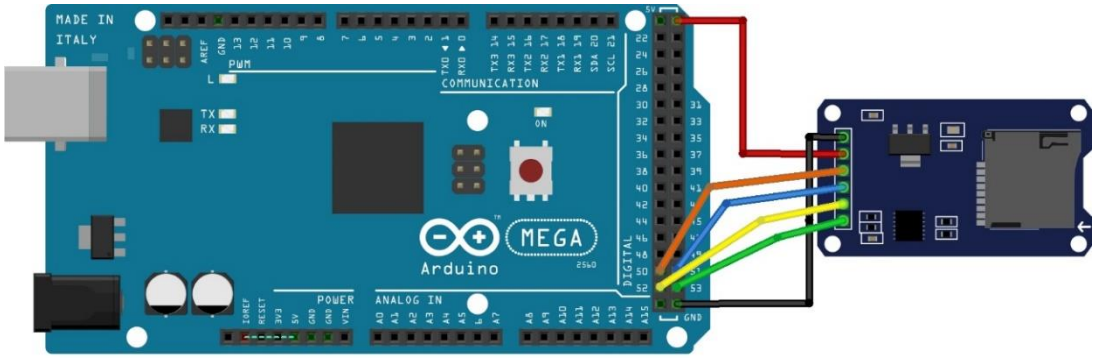
1.1.- DRV8825 Stepper Motor Controller Wiring[1]



1.2.- HC-06 Bluetooth Module Wiring

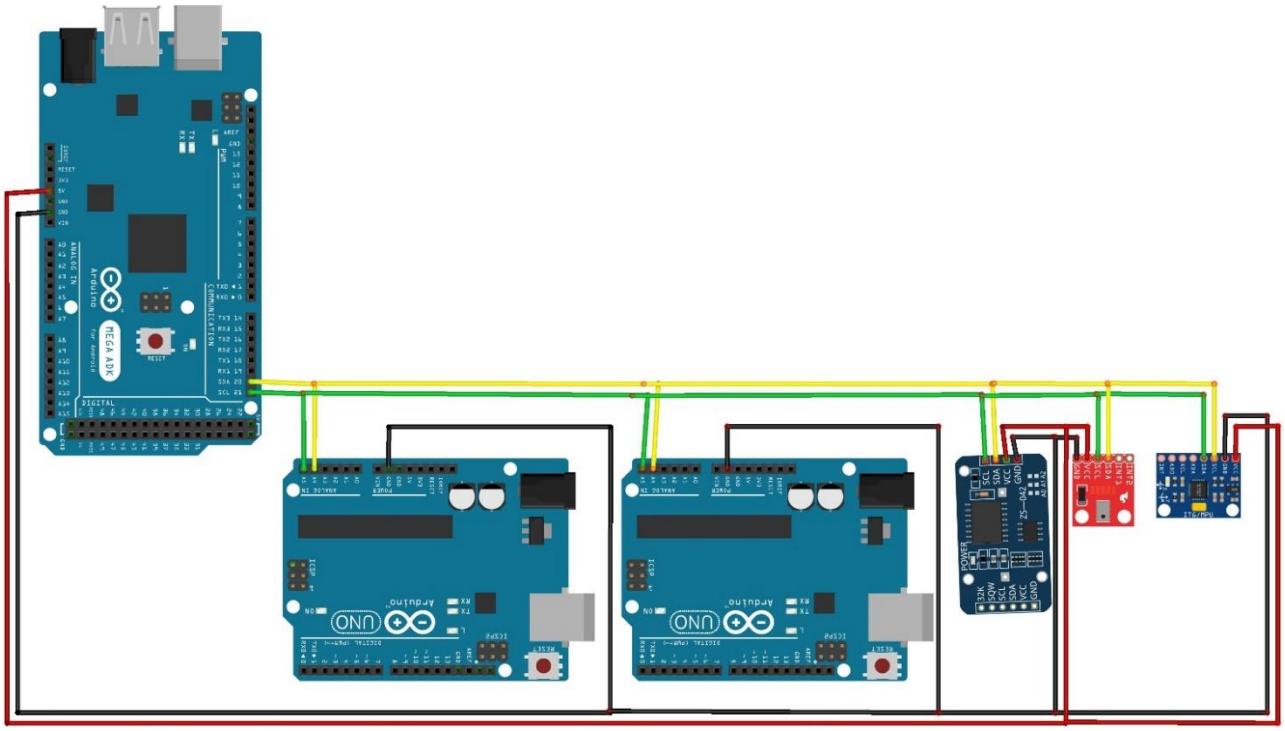


1.3.- SD Adapter Module Wiring



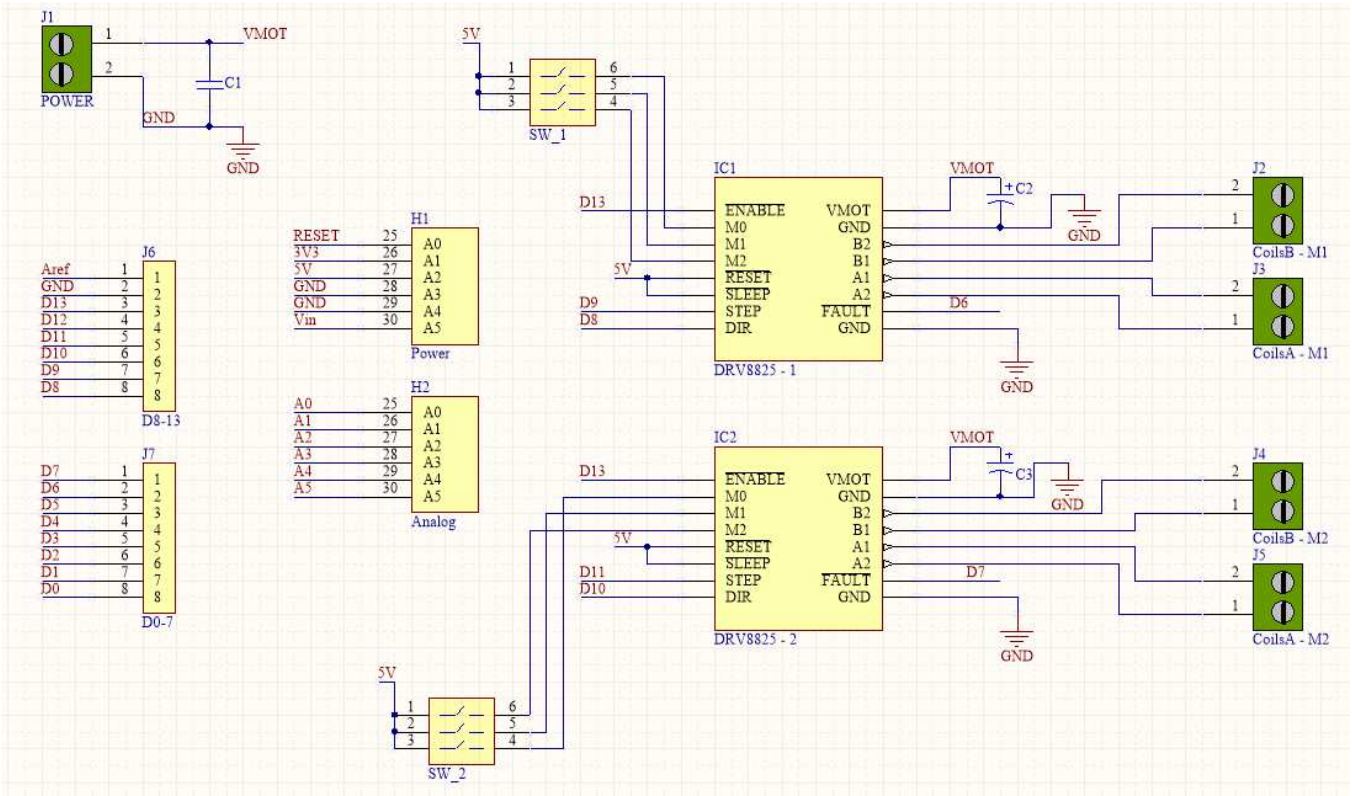
[1] Anonymous, “DRV8825 Stepper Motor Driver Carrier, High Current, Item #2133”, Pololu Stepper Motor Drivers, https://www.pololu.com/product/2133

1.4.- I2C Bus Wiring Schematics

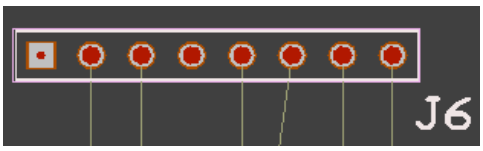


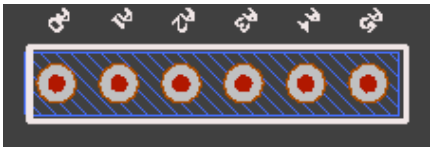
2.- Arduino Shield PCB Schematics

2.1.- Controller Shield Electric Schematic

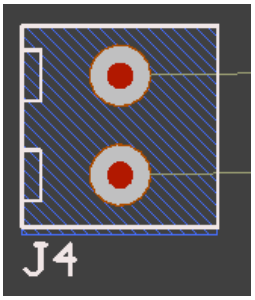


2.2.- PCB Socket Footprint

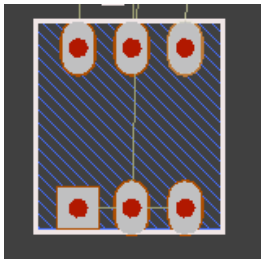




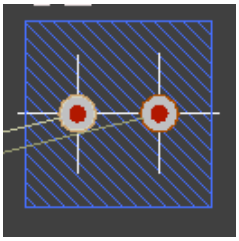
2.3.- PCB Connection Block Footprint



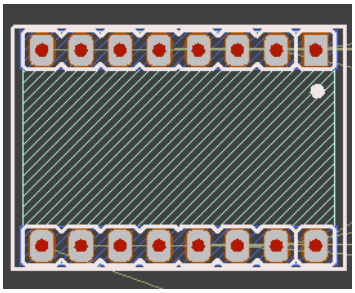
2.4.- 3 DIP Switch PCB Footprint



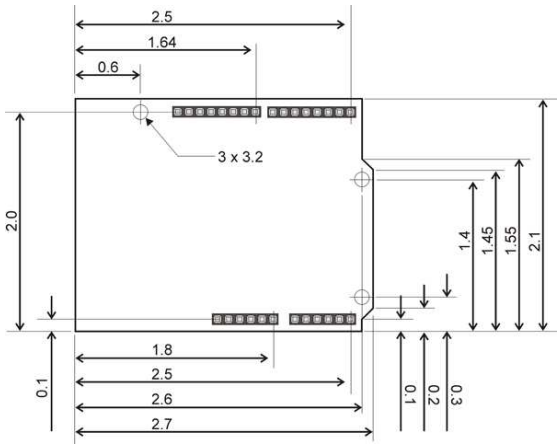
2.5.- Decoupling Capacitor PCB Footprint



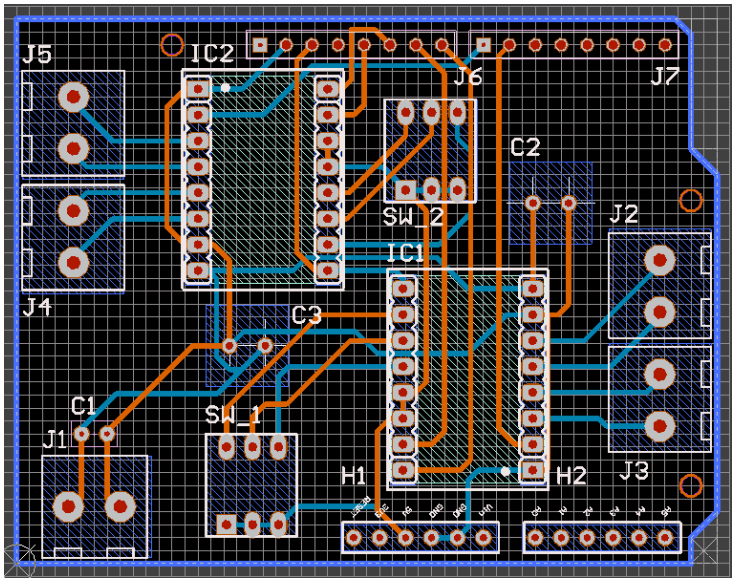
2.6.- DRV8825 PCB Footprint



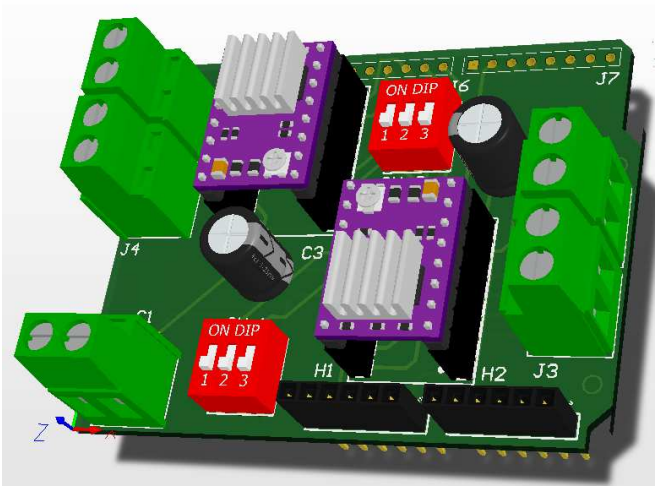
2.7.- PCB Shield Dimensions Diagram



2.8.- PCB Final Layout and Routing Schematics

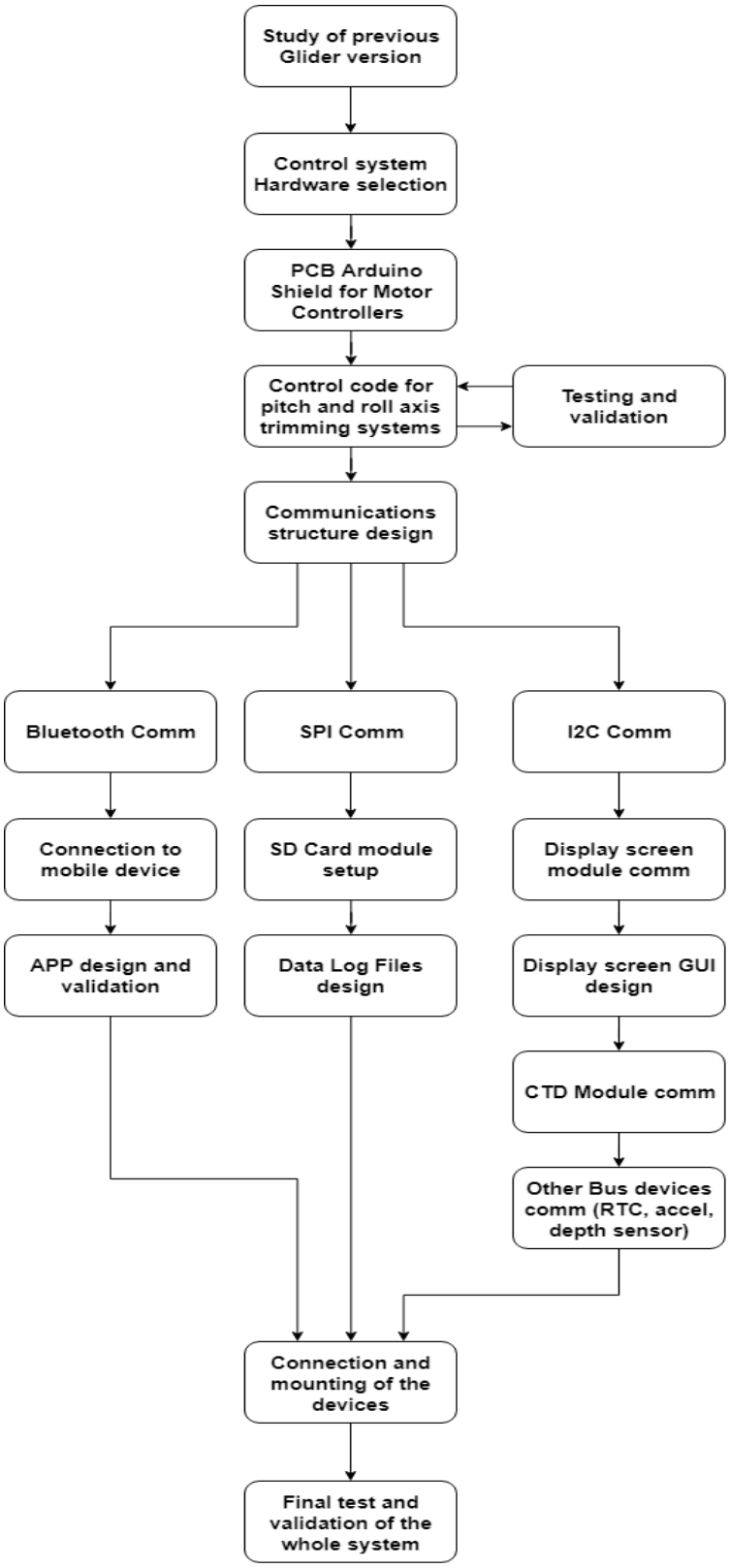


2.9.- PCB 3D Model

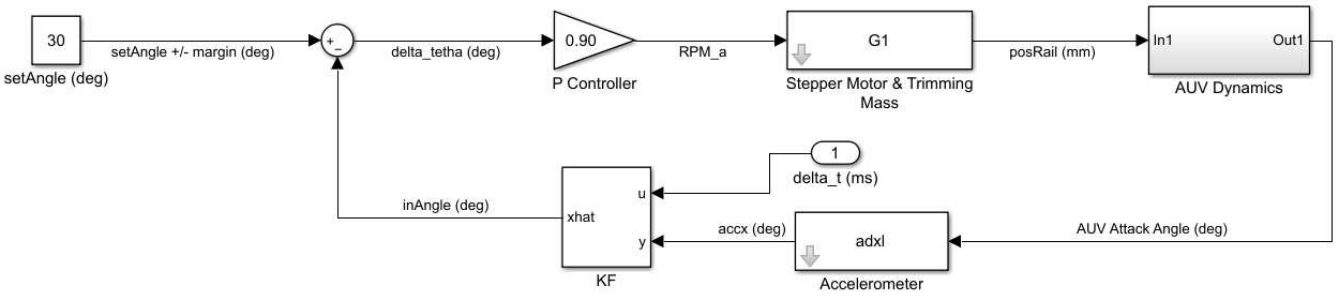


3.- Diagrams

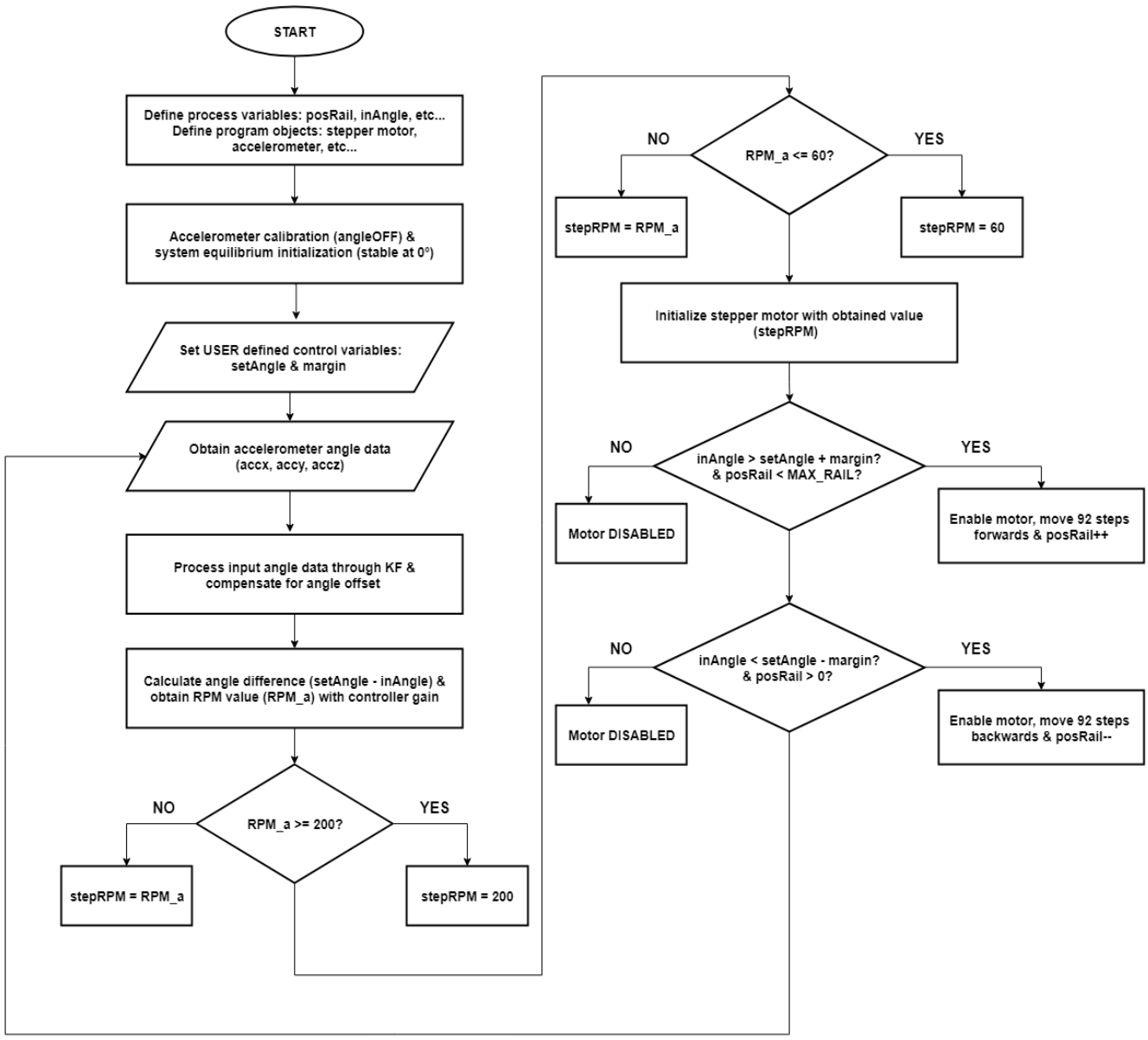
3.1.- Planning Diagram



3.2.- AUV Trimming Control System Diagram



3.3.- Attitude Control System Flow Chart



DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

ESCUELA TÉCNICA SUEPERIOR DE INGENIERÍA DEL DISEÑO

UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEVELOPMENT OF THE CONTROL ELECTRONICS FOR THE NAVIGATION OF AN UNMANNED SUBMARINE WITH ARDUINO

5. ANNEXES

TRABAJO DE FIN DE GRADO:

Grado en Ingeniería Electrónica Industrial y Automática

Autor: Fco. Javier Pérez Villaplana

Director: Jose Vicente Busquets Mataix

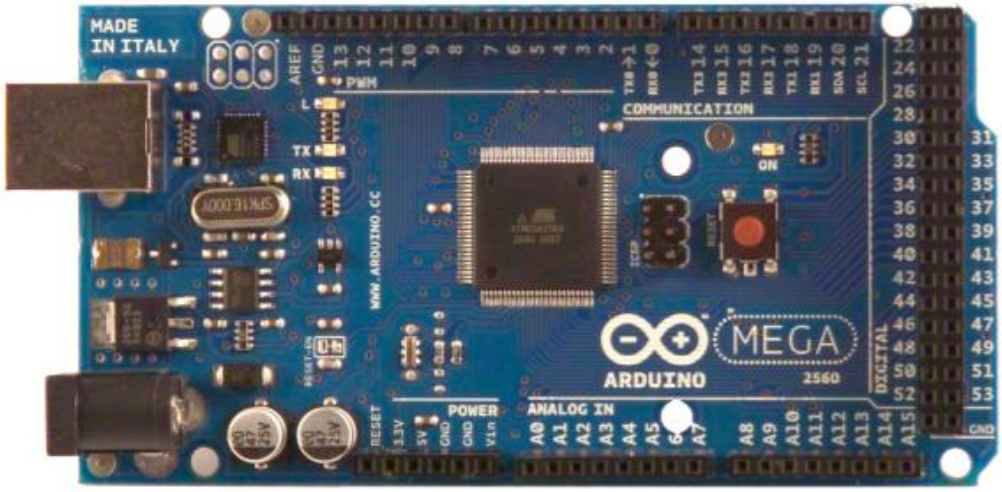
Codirector: Javier Busquets Mataix

VALENCIA, ABRIL 2019

Index

1.- Arduino MEGA Datasheet (ATmega2560)[1]

Arduino MEGA 2560



CE

Product Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

Index

[1] http://www.mantech.co.za/datasheets/products/A000047.pdf

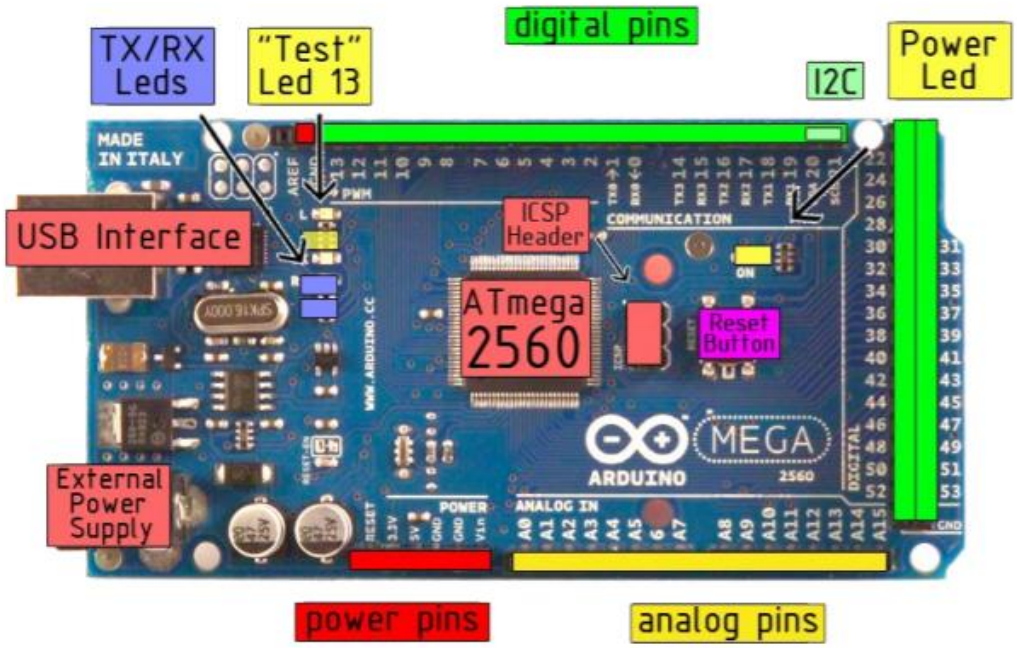
Technical Specification

EAGLE files: arduino-mega2560-reference-design.zip Schematic: arduino-mega2560-schematic.pdf

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

the board



Power

The Arduino Mega2560 can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **I^2C: 20 (SDA) and 21 (SCL).** Support I^2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I^2C pins on the Duemilanove.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega8U2 on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Mega's digital pins.

The ATmega2560 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website for details. To use the SPI communication, please see the ATmega2560 datasheet.

Programming

The Arduino Mega2560 can be programmed with the Arduino software (download). For details, see the reference and tutorials.

The Atmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Mega contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Mega has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega is designed to be compatible with most shields designed for the Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega and Duemilanove / Diecimila. **Please note that I^2C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).**

How to use Arduino

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).

Arduino is a cross-platoform program. You'll have to follow different instructions for your personal OS. Check on the Arduino site for the latest instructions. *http://arduino.cc/en/Guide/HomePage*

Linux Install Windows Install Mac Install

Once you have downloaded/unzipped the arduino IDE, you can Plug the Arduino to your PC via USB cable.

Blink led

Now you're actually ready to "burn" your first program on the arduino board. To select "blink led", the physical translation of the well known programming "hello world", select

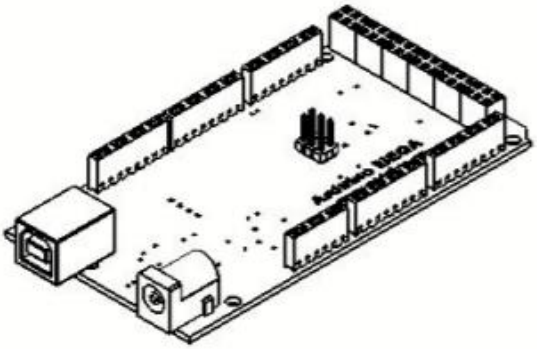
File>Sketchbook>
Arduino-0017>Examples>
Digital>Blink

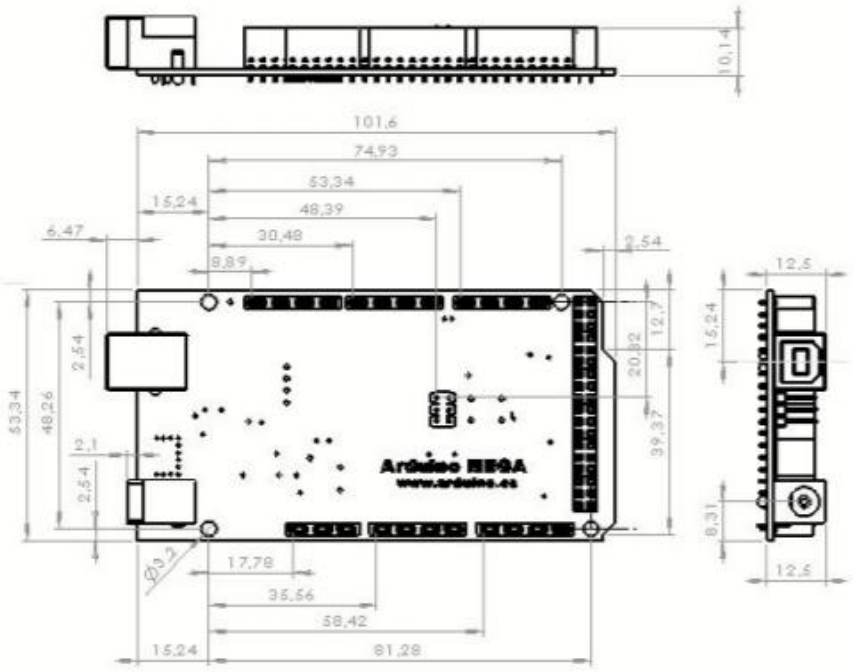
Once you have your skecth you'll see something very close to the screenshot on the right.

In **Tools>Board** select MEGA

Now you have to go to
Tools>SerialPort
and select the right serial port, the one arduino is attached to.

Dimensioned Drawing

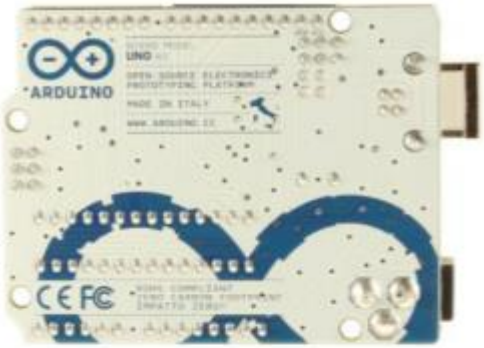




2.- Arduino UNO Datasheet (ATmega328)[2]

Arduino Uno

Arduino Uno R3 Front



Arduino Uno R3 Back

Arduino Uno R2 Front

Arduino Uno SMD

Arduino Uno Front

Arduino Uno Back

Overview

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.
The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.
Revision 2 of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode.
Revision 3 of the board has the following new features:

- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for future purposes.
- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V

[2] https://www.farnell.com/datasheets/1682209.pdf

Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Schematic & Reference Design

EAGLE files: arduino-uno-Rev3-reference-design.zip (NOTE: works with Eagle 6.0 and newer)
Schematic: arduino-uno-Rev3-schematic.pdf
Note: The Arduino reference design can use an Atmega8, 168, or 328, Current models use an ATmega328, but an Atmega8 is shown in the schematic for reference. The pin configuration is identical on all three processors.

Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.
External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.
The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.
The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.**This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.

- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the SPI library.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- **TWI: A4 or SDA pin and A5 or SCL pin.** Support TWI communication using the Wire library.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).
A SoftwareSerial library allows for serial communication on any of the Uno's digital pins.
The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

Programming

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the reference and tutorials.
The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).
You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.
The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.
The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

3.- DRV8825 Stepper Motor Controller Datasheet[3]

TEXAS INSTRUMENTS

DRV8825

SLVSA73F – APRIL 2010 – REVISED JULY 2014

DRV8825 Stepper Motor Controller IC

1 Features

- PWM Microstepping Stepper Motor Driver
  - Built-In Microstepping Indexer
  - Up to 1/32 Microstepping
- Multiple Decay Modes
  - Mixed Decay
  - Slow Decay
  - Fast Decay
- 8.2-V to 45-V Operating Supply Voltage Range
- 2.5-A Maximum Drive Current at 24 V and T_A = 25°C
- Simple STEP/DIR Interface
- Low Current Sleep Mode
- Built-In 3.3-V Reference Output
- Small Package and Footprint
- Protection Features
  - Overcurrent Protection (OCP)
  - Thermal Shutdown (TSD)
  - VM Undervoltage Lockout (UVLO)
  - Fault Condition Indication Pin (nFAULT)

2 Applications

- Automatic Teller Machines
- Money Handling Machines
- Video Security Cameras
- Printers
- Scanners
- Office Automation Machines
- Gaming Machines
- Factory Automation
- Robotics

3 Description

The DRV8825 provides an integrated motor driver solution for printers, scanners, and other automated equipment applications. The device has two H-bridge drivers and a microstepping indexer, and is intended to drive a bipolar stepper motor. The output driver block consists of N-channel power MOSFET's configured as full H-bridges to drive the motor windings. The DRV8825 is capable of driving up to 2.5 A of current from each output (with proper heat sinking, at 24 V and 25°C).

A simple STEP/DIR interface allows easy interfacing to controller circuits. Mode pins allow for configuration of the motor in full-step up to 1/32-step modes. Decay mode is configurable so that slow decay, fast decay, or mixed decay can be used. A low-power sleep mode is provided which shuts down internal circuitry to achieve very low quiescent current draw. This sleep mode can be set using a dedicated nSLEEP pin.

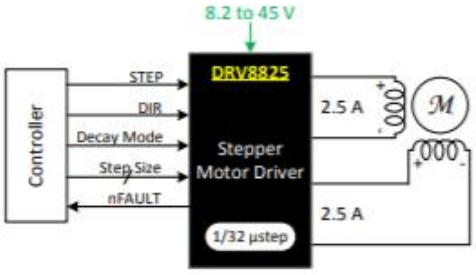
Internal shutdown functions are provided for overcurrent, short circuit, under voltage lockout and over temperature. Fault conditions are indicated via the nFAULT pin.

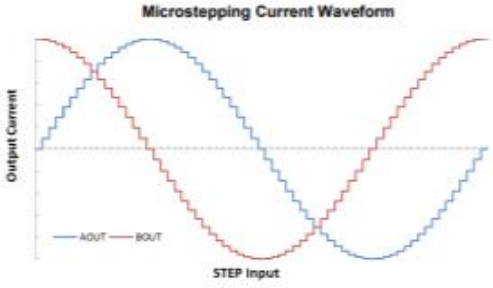
Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
DRV8825	HTSSOP (28)	9.70 mm × 6.40 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

4 Simplified Schematic





An IMPORTANT NOTICE at the end of this data sheet addresses availability, warranty, changes, use in safety-critical applications, intellectual property matters and other important disclaimers. PRODUCTION DATA.

[3] https://www.pololu.com/file/0J590/drv8825.pdf

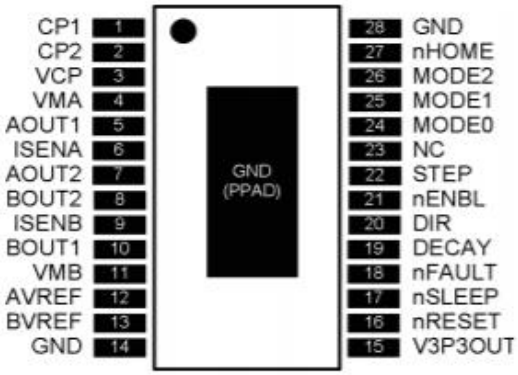
Table of Contents

5 Revision History

Changes from Revision E (August 2013) to Revision F — **Page**

- Added new sections and reordered data sheet to fit new TI flow ... 1
- Updated pin descriptions ... 3
- Added power supply ramp rate and updated ISENSEx pin voltage in *Absolute Maximum Ratings* ... 4
- Updated V_{IL} voltage minimum and typical in *Electrical Characteristics* ... 6
- Updated I_{IN} and t_{DEG} in *Electrical Characteristics* ... 6

6 Pin Configuration and Functions



Pin Functions

PIN NAME	PIN NO.	I/O(1)	DESCRIPTION	EXTERNAL COMPONENTS OR CONNECTIONS
POWER AND GROUND				
CP1	1	I/O	Charge pump flying capacitor	Connect a 0.01-µF 50-V capacitor between CP1 and CP2.
CP2	2	I/O	Charge pump flying capacitor	
GND	14, 28	—	Device ground	
VCP	3	I/O	High-side gate drive voltage	Connect a 0.1-µF 16-V ceramic capacitor and a 1-MΩ resistor to VM.
VMA	4	—	Bridge A power supply	Connect to motor supply (8.2 to 45 V). Both pins must be connected to the same supply, bypassed with a 0.1-µF capacitor to GND, and connected to appropriate bulk capacitance.
VMB	11	—	Bridge B power supply	
V3P3OUT	15	O	3.3-V regulator output	Bypass to GND with a 0.47-µF 6.3-V ceramic capacitor. Can be used to supply VREF.
CONTROL				
AVREF	12	I	Bridge A current set reference input	Reference voltage for winding current set. Normally AVREF and BVREF are connected to the same voltage. Can be connected to V3P3OUT.
BVREF	13	I	Bridge B current set reference input	
DECAY	19	I	Decay mode	Low = slow decay, open = mixed decay, high = fast decay. Internal pulldown and pullup.
DIR	20	I	Direction input	Level sets the direction of stepping. Internal pulldown.
MODE0	24	I	Microstep mode 0	MODE0 through MODE2 set the step mode - full, 1/2, 1/4, 1/8/ 1/16, or 1/32 step. Internal pulldown.
MODE1	25	I	Microstep mode 1	
MODE2	26	I	Microstep mode 2	
NC	23	—	No connect	Leave this pin unconnected.
nENBL	21	I	Enable input	Logic high to disable device outputs and indexer operation, logic low to enable. Internal pulldown.
nRESET	16	I	Reset input	Active-low reset input initializes the indexer logic and disables the H-bridge outputs. Internal pulldown.
nSLEEP	17	I	Sleep mode input	Logic high to enable device, logic low to enter low-power sleep mode. Internal pulldown.
STEP	22	I	Step input	Rising edge causes the indexer to move one step. Internal pulldown.
STATUS				
nFAULT	18	OD	Fault	Logic low when in fault condition (overtemp, overcurrent)

(1) Directions: I = input, O = output, OD = open-drain output, IO = input/output

Pin Functions (continued)

PIN		I/O[1]	DESCRIPTION	EXTERNAL COMPONENTS OR CONNECTIONS
NAME	NO.			
nHOME	27	OD	Home position	Logic low when at home state of step table
OUTPUT				
AOUT1	5	O	Bridge A output 1	Connect to bipolar stepper motor winding A. Positive current is AOUT1 → AOUT2
AOUT2	7	O	Bridge A output 2	
BOUT1	10	O	Bridge B output 1	Connect to bipolar stepper motor winding B. Positive current is BOUT1 → BOUT2
BOUT2	8	O	Bridge B output 2	
ISENA	6	I/O	Bridge A ground / Isense	Connect to current sense resistor for bridge A.
ISENB	9	I/O	Bridge B ground / Isense	Connect to current sense resistor for bridge B.

7 Specifications

7.1 Absolute Maximum Ratings[1][2]

		MIN	MAX	UNIT
$V_{(VMx)}$	Power supply voltage	–0.3	47	V
	Power supply ramp rate		1	V/µs
	Digital pin voltage	–0.5	7	V
$V_{(xVREF)}$	Input voltage	–0.3	4	V
	ISENSEx pin voltage[3]	–0.8	0.8	V
	Peak motor drive output current, t < 1 µs		Internally limited	A
	Continuous motor drive output current[4]	0	2.5	A
	Continuous total power dissipation		See *Thermal Information*	
T_J	Operating junction temperature range	–40	150	°C

(1) Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions* is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.
(2) All voltage values are with respect to network ground terminal.
(3) Transients of ±1 V for less than 25 ns are acceptable
(4) Power dissipation and thermal limits must be observed.

7.2 Handling Ratings

			MIN	MAX	UNIT
T_{stg}	Storage temperature range		–60	150	°C
$V_{(ESD)}$	Electrostatic discharge	Human body model (HBM), per ANSI/ESDA/JEDEC JS-001, all pins[1]	–2000	2000	V
		Charged device model (CDM), per JEDEC specification JESD22-C101, all pins[2]	–500	500	

(1) JEDEC document JEP155 states that 500-V HBM allows safe manufacturing with a standard ESD control process.
(2) JEDEC document JEP157 states that 250-V CDM allows safe manufacturing with a standard ESD control process.

7.3 Recommended Operating Conditions

		MIN	NOM	MAX	UNIT
$V_{(VMx)}$	Motor power supply voltage range[1]	8.2		45	V
$V_{(VREF)}$	VREF input voltage[2]	1		3.5	V
I_{V3P3}	V3P3OUT load current	0		1	mA

(1) All V_M pins must be connected to the same supply voltage.
(2) Operational at VREF between 0 to 1 V, but accuracy is degraded.

7.4 Thermal Information

THERMAL METRIC[1]		DRV8825 PWP 28 PINS	UNIT
$R_{\theta JA}$	Junction-to-ambient thermal resistance[2]	31.6	°C/W
$R_{\theta JC(top)}$	Junction-to-case (top) thermal resistance[3]	15.9	
$R_{\theta JB}$	Junction-to-board thermal resistance[4]	5.6	
Ψ_{JT}	Junction-to-top characterization parameter[5]	0.2	
Ψ_{JB}	Junction-to-board characterization parameter[6]	5.5	
$R_{\theta JC(bot)}$	Junction-to-case (bottom) thermal resistance[7]	1.4	

(1) For more information about traditional and new thermal metrics, see the *IC Package Thermal Metrics* application report, SPRA953.
(2) The junction-to-ambient thermal resistance under natural convection is obtained in a simulation on a JEDEC-standard, high-K board, as specified in JESD51-7, in an environment described in JESD51-2a.
(3) The junction-to-case (top) thermal resistance is obtained by simulating a cold plate test on the package top. No specific JEDEC-standard test exists, but a close description can be found in the ANSI SEMI standard G30-88.
(4) The junction-to-board thermal resistance is obtained by simulating in an environment with a ring cold plate fixture to control the PCB temperature, as described in JESD51-8.
(5) The junction-to-top characterization parameter, ψ_{JT}, estimates the junction temperature of a device in a real system and is extracted from the simulation data for obtaining θ_{JA}, using a procedure described in JESD51-2a (sections 6 and 7).
(6) The junction-to-board characterization parameter, ψ_{JB}, estimates the junction temperature of a device in a real system and is extracted from the simulation data for obtaining θ_{JA} , using a procedure described in JESD51-2a (sections 6 and 7).
(7) The junction-to-case (bottom) thermal resistance is obtained by simulating a cold plate test on the exposed (power) pad. No specific JEDEC standard test exists, but a close description can be found in the ANSI SEMI standard G30-88.

7.5 Electrical Characteristics

over operating free-air temperature range of –40°C to 85°C (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
POWER SUPPLIES						
I_{VM}	VM operating supply current	$V_{(VMx)}$ = 24 V		5	8	mA
I_{VMQ}	VM sleep mode supply current	$V_{(VMx)}$ = 24 V		10	20	µA
V3P3OUT REGULATOR						
V_{3P3}	V3P3OUT voltage	IOUT = 0 to 1 mA	3.2	3.3	3.4	V
LOGIC-LEVEL INPUTS						
V_{IL}	Input low voltage		0		0.7	V
V_{IH}	Input high voltage		2.2		5.25	V
V_{HYS}	Input hysteresis		0.3	0.45	0.6	V
I_{IL}	Input low current	VIN = 0	–20		20	µA
I_{IH}	Input high current	VIN = 3.3 V			100	µA
R_{PD}	Internal pulldown resistance			100		kΩ
nHOME, nFAULT OUTPUTS (OPEN-DRAIN OUTPUTS)						
V_{OL}	Output low voltage	I_O = 5 mA			0.5	V
I_{OH}	Output high leakage current	V_O = 3.3 V			1	µA
DECAY INPUT						
V_{IL}	Input low threshold voltage	For slow decay mode			0.8	V
V_{IH}	Input high threshold voltage	For fast decay mode	2			V
I_{IN}	Input current		–40		40	µA
R_{PU}	Internal pullup resistance (to 3.3 V)			130		kΩ
R_{PD}	Internal pulldown resistance			80		kΩ
H-BRIDGE FETS						
$R_{DS(ON)}$	HS FET on resistance	$V_{(VMx)}$ = 24 V, I_O = 1 A, T_J = 25°C		0.2		Ω
		$V_{(VMx)}$ = 24 V, I_O = 1 A, T_J = 85°C		0.25	0.32	
	LS FET on resistance	$V_{(VMx)}$ = 24 V, I_O = 1 A, T_J = 25°C		0.2		
		$V_{(VMx)}$ = 24 V, I_O = 1 A, T_J = 85°C		0.25	0.32	
I_{OFF}	Off-state leakage current		–20		20	µA
MOTOR DRIVER						
f_{PWM}	Internal current control PWM frequency			30		kHz
t_{BLANK}	Current sense blanking time			4		µs
t_R	Rise time		30		200	ns
t_F	Fall time		30		200	ns
PROTECTION CIRCUITS						
V_{UVLO}	VM undervoltage lockout voltage	$V_{(VMx)}$ rising		7.8	8.2	V
I_{OCP}	Overcurrent protection trip level		3			A
t_{DEG}	Overcurrent deglitch time			3		µs
t_{TSD}	Thermal shutdown temperature	Die temperature	150	160	180	°C
CURRENT CONTROL						
I_{REF}	xVREF input current	$V_{(xVREF)}$ = 3.3 V	–3		3	µA
V_{TRIP}	xISENSE trip voltage	$V_{(xVREF)}$ = 3.3 V, 100% current setting	635	660	685	mV
ΔI_{TRIP}	Current trip accuracy (relative to programmed value)	$V_{(xVREF)}$ = 3.3 V, 5% current setting	–25%		25%	
		$V_{(xVREF)}$ = 3.3 V, 10% to 34% current setting	–15%		15%	
		$V_{(xVREF)}$ = 3.3 V, 38% to 67% current setting	–10%		10%	
		$V_{(xVREF)}$ = 3.3 V, 71% to 100% current setting	–5%		5%	
A_{ISENSE}	Current sense amplifier gain	Reference only		5		V/V

7.6 Timing Requirements

			MIN	MAX	UNIT
1	f_{STEP}	Step frequency		250	kHz
2	$t_{WH(STEP)}$	Pulse duration, STEP high	1.9		µs
3	$t_{WL(STEP)}$	Pulse duration, STEP low	1.9		µs
4	$t_{SU(STEP)}$	Setup time, command before STEP rising	650		ns
5	$t_{H(STEP)}$	Hold time, command after STEP rising	650		ns
6	t_{ENBL}	Enable time, nENBL active to STEP	650		ns
7	t_{WAKE}	Wakeup time, nSLEEP inactive high to STEP input accepted		1.7	ms

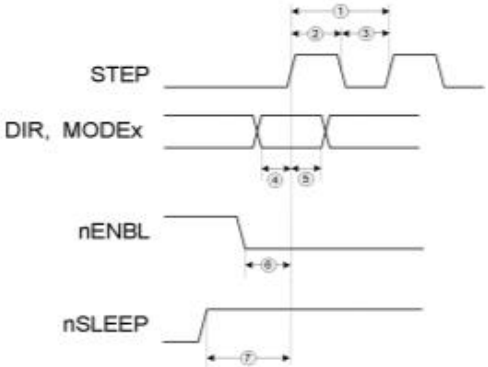


Figure 1. Timing Diagram

7.7 Typical Characteristics

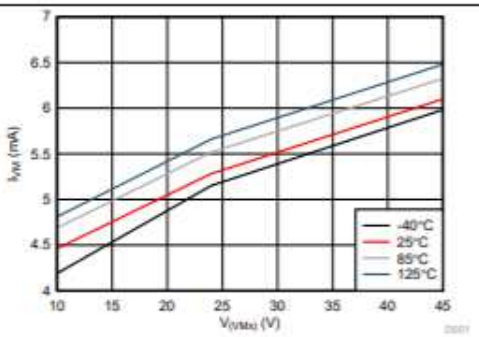


Figure 2. I_{VMx} vs $V_{(VMx)}$

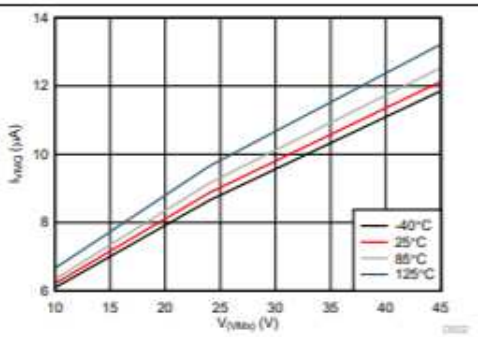


Figure 3. I_{VMxQ} vs $V_{(VMx)}$

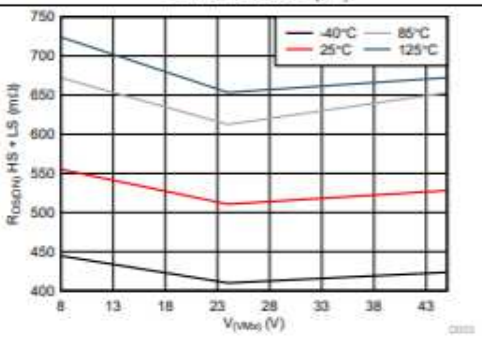


Figure 4. $R_{DS(ON)}$ vs $V_{(VMx)}$

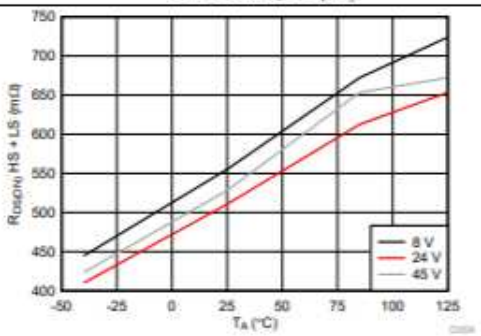


Figure 5. $R_{DS(ON)}$ vs Temperature

8 Detailed Description

8.1 Overview

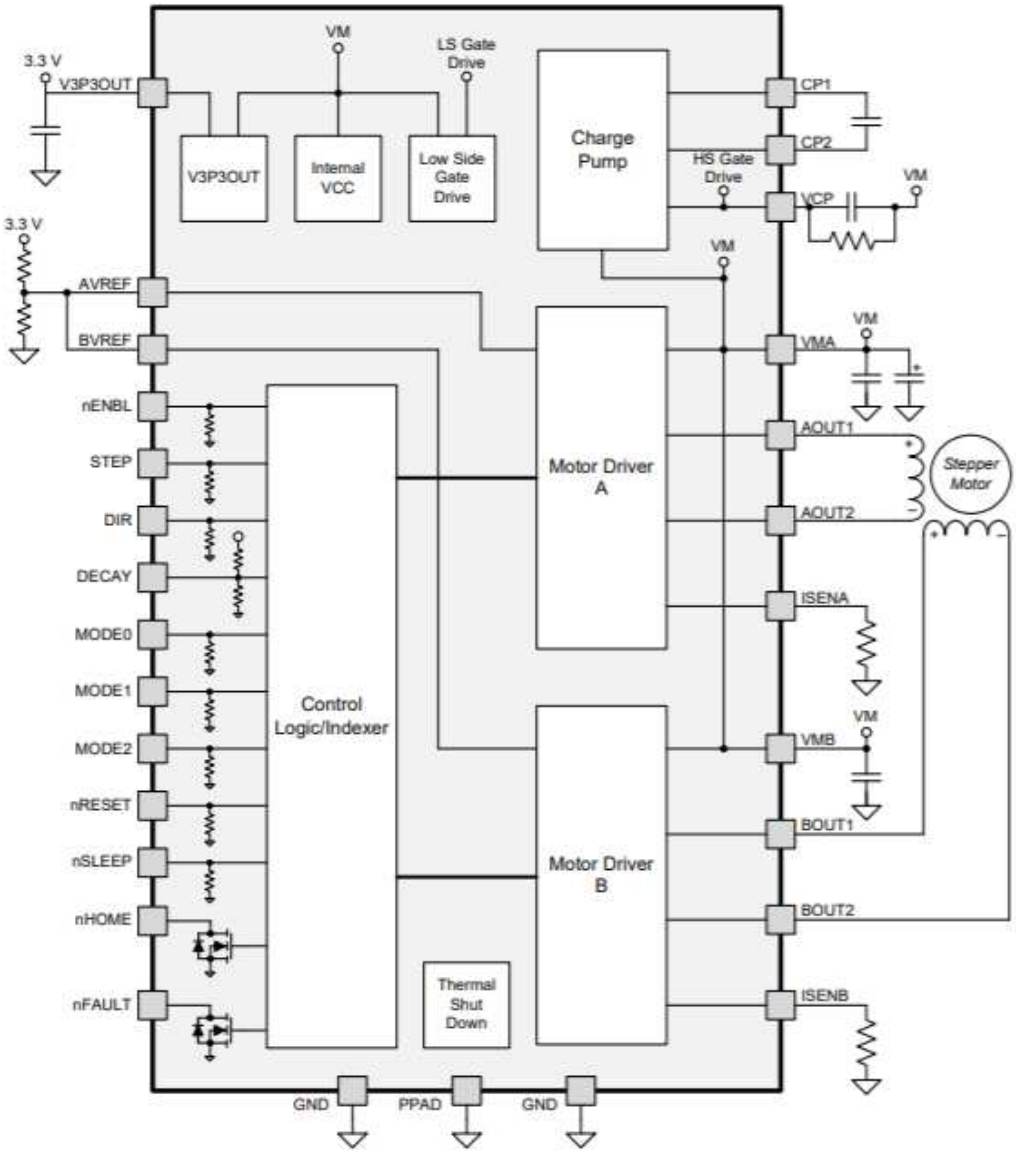
The DRV8825 is an integrated motor driver solution for bipolar stepper motors. The device integrates two NMOS H-bridges, current sense, regulation circuitry, and a microstepping indexer. The DRV8825 can be powered with a supply voltage between 8.2 and 45 V and is capable of providing an output current up to 2.5 A full-scale.

A simple STEP/DIR interface allows for easy interfacing to the controller circuit. The internal indexer is able to execute high-accuracy microstepping without requiring the processor to control the current level.

The current regulation is highly configurable, with three decay modes of operation. Depending on the application requirements, the user can select fast, slow, and mixed decay.

A low-power sleep mode is included which allows the system to save power when not driving the motor.

8.2 Functional Block Diagram



8.3 Feature Description

8.3.1 PWM Motor Drivers

The DRV8825 contains two H-bridge motor drivers with current-control PWM circuitry. Figure 6 shows a block diagram of the motor control circuitry.

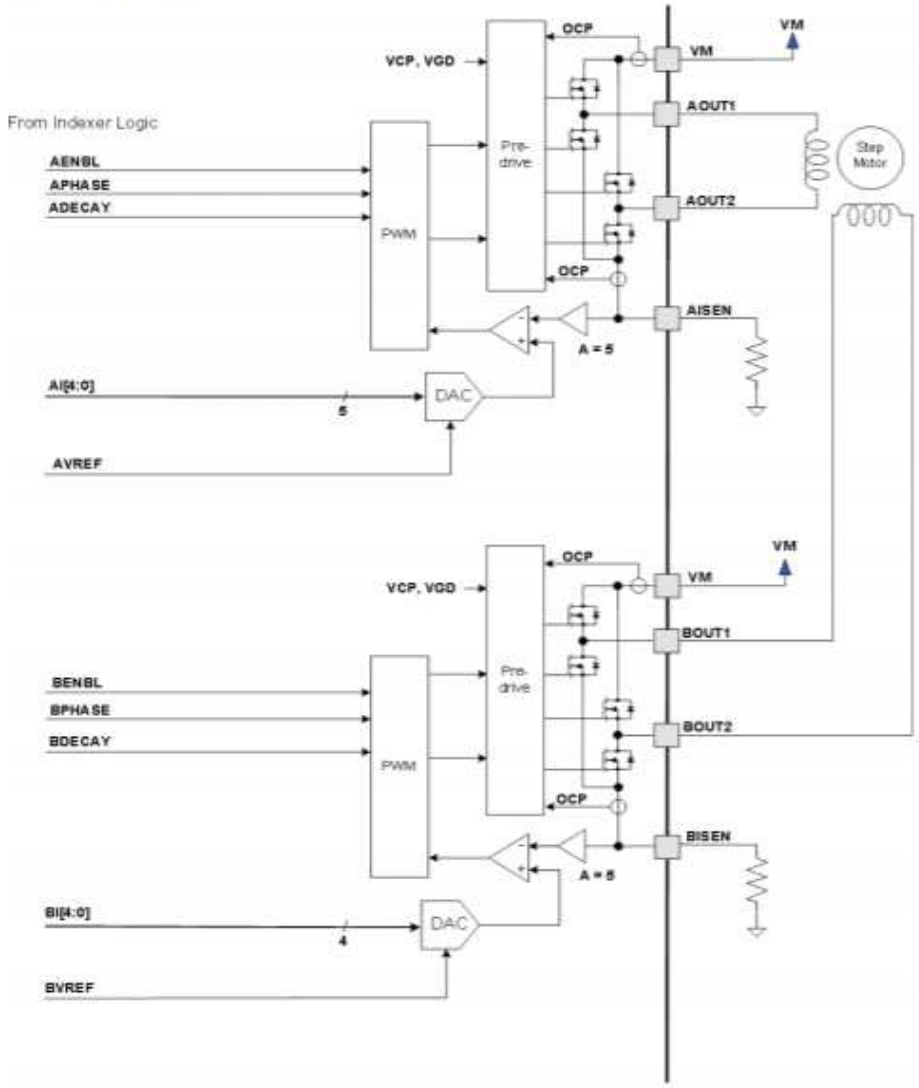


Figure 6. Motor Control Circuitry

Note that there are multiple VM motor power supply pins. All VM pins must be connected together to the motor supply voltage.

Feature Description (continued)

8.3.2 Current Regulation

The current through the motor windings is regulated by a fixed-frequency PWM current regulation, or current chopping. When an H-bridge is enabled, current rises through the winding at a rate dependent on the DC voltage and inductance of the winding. Once the current hits the current chopping threshold, the bridge disables the current until the beginning of the next PWM cycle.

In stepping motors, current regulation is used to vary the current in the two windings in a semi-sinusoidal fashion to provide smooth motion.

The PWM chopping current is set by a comparator which compares the voltage across a current sense resistor connected to the xISEN pins, multiplied by a factor of 5, with a reference voltage. The reference voltage is input from the xVREF pins.

The full-scale (100%) chopping current is calculated in Equation 1.

$$I_{CHOP} = \frac{V_{(xREF)}}{5 \times R_{ISENSE}} \quad (1)$$

Example:

If a 0.25-Ω sense resistor is used and the VREFx pin is 2.5 V, the full-scale (100%) chopping current will be 2.5 V / (5 x 0.25 Ω) = 2 A.

The reference voltage is scaled by an internal DAC that allows fractional stepping of a bipolar stepper motor, as described in the microstepping indexer section below.

8.3.3 Decay Mode

During PWM current chopping, the H-bridge is enabled to drive current through the motor winding until the PWM current chopping threshold is reached. This is shown in Figure 7 as case 1. The current flow direction shown indicates positive current flow.

Once the chopping current threshold is reached, the H-bridge can operate in two different states, fast decay or slow decay.

In fast decay mode, once the PWM chopping current level has been reached, the H-bridge reverses state to allow winding current to flow in a reverse direction. As the winding current approaches 0, the bridge is disabled to prevent any reverse current flow. Fast decay mode is shown in Figure 7 as case 2.

In slow decay mode, winding current is recirculated by enabling both of the low-side FETs in the bridge. This is shown in Figure 7 as case 3.

Feature Description (continued)

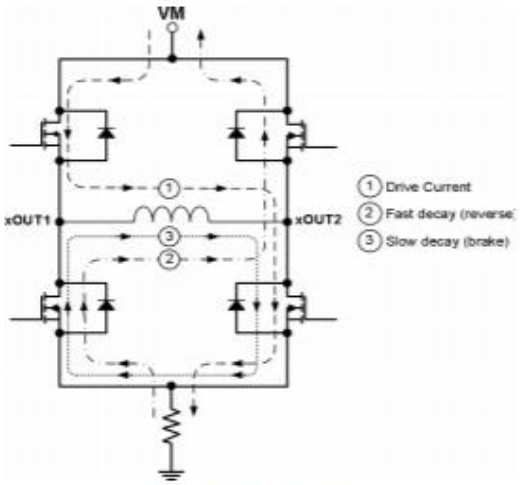


Figure 7. Decay Mode

The DRV8825 supports fast decay, slow decay and a mixed decay mode. Slow, fast, or mixed decay mode is selected by the state of the DECAY pin; logic low selects slow decay, open selects mixed decay operation, and logic high sets fast decay mode. The DECAY pin has both an internal pullup resistor of approximately 130 kΩ and an internal pulldown resistor of approximately 80 kΩ. This sets the mixed decay mode if the pin is left open or undriven.

Mixed decay mode begins as fast decay, but at a fixed period of time (75% of the PWM cycle) switches to slow decay mode for the remainder of the fixed PWM period. This occurs only if the current through the winding is decreasing (per the indexer step table); if the current is increasing, then slow decay is used.

8.3.4 Blanking Time

After the current is enabled in an H-bridge, the voltage on the xISEN pin is ignored for a fixed period of time before enabling the current sense circuitry. This blanking time is fixed at 3.75 µs. Note that the blanking time also sets the minimum on time of the PWM.

8.3.5 Microstepping Indexer

Built-in indexer logic in the DRV8825 allows a number of different stepping configurations. The MODE0 through MODE2 pins are used to configure the stepping format as shown in Table 1.

Table 1. Stepping Format

MODE2	MODE1	MODE0	STEP MODE
0	0	0	Full step (2-phase excitation) with 71% current
0	0	1	1/2 step (1-2 phase excitation)
0	1	0	1/4 step (W1-2 phase excitation)
0	1	1	8 microsteps/step
1	0	0	16 microsteps/step
1	0	1	32 microsteps/step
1	1	0	32 microsteps/step
1	1	1	32 microsteps/step

Table 2 shows the relative current and step directions for different settings of MODEx. At each rising edge of the STEP input, the indexer travels to the next state in the table. The direction is shown with the DIR pin high; if the DIR pin is low the sequence is reversed. Positive current is defined as xOUT1 = positive with respect to xOUT2.

Note that if the step mode is changed while stepping, the indexer will advance to the next valid state for the new MODEx setting at the rising edge of STEP.

The home state is 45°. This state is entered at power-up or application of nRESET. This is shown in Table 2 by the shaded cells. The logic inputs DIR, STEP, nRESET, and MODEx have internal pulldown resistors of 100 kΩ.

Table 2. Relative Current and Step Directions

1/32 STEP	1/16 STEP	1/8 STEP	1/4 STEP	1/2 STEP	FULL STEP 70%	WINDING CURRENT A	WINDING CURRENT B	ELECTRICAL ANGLE
1	1	1	1	1		100%	0%	0
2						100%	5%	3
3	2					100%	10%	6
4						99%	15%	8
5	3	2				98%	20%	11
6						97%	24%	14
7	4					96%	29%	17
8						94%	34%	20
9	5	3	2			92%	38%	23
10						90%	43%	25
11	6					88%	47%	28
12						86%	51%	31
13	7	4				83%	56%	34
14						80%	60%	37
15	8					77%	63%	39
16						74%	67%	42
17	9	5	3	2	1	71%	71%	45
18						67%	74%	48
19	10					63%	77%	51
20						60%	80%	53
21	11	6				56%	83%	56
22						51%	86%	59
23	12					47%	88%	62
24						43%	90%	65
25	13	7	4			38%	92%	68
26						34%	94%	70
27	14					29%	96%	73
28						24%	97%	76
29	15	8				20%	98%	79
30						15%	99%	82
31	16					10%	100%	84
32						5%	100%	87
33	17	9	5	3		0%	100%	90
34						–5%	100%	93
35	18					–10%	100%	96
36						–15%	99%	98
37	19	10				–20%	98%	101
38						–24%	97%	104
39	20					–29%	96%	107

Table 2. Relative Current and Step Directions (continued)

1/32 STEP	1/16 STEP	1/8 STEP	1/4 STEP	1/2 STEP	FULL STEP 70%	WINDING CURRENT A	WINDING CURRENT B	ELECTRICAL ANGLE
40						–34%	94%	110
41	21	11	6			–38%	92%	113
42						–43%	90%	115
43	22					–47%	88%	118
44						–51%	86%	121
45	23	12				–56%	83%	124
46						–60%	80%	127
47	24					–63%	77%	129
48						–67%	74%	132
49	25	13	7	4	2	–71%	71%	135
50						–74%	67%	138
51	26					–77%	63%	141
52						–80%	60%	143
53	27	14				–83%	56%	146
54						–86%	51%	149
55	28					–88%	47%	152
56						–90%	43%	155
57	29	15	8			–92%	38%	158
58						–94%	34%	160
59	30					–96%	29%	163
60						–97%	24%	166
61	31	16				–98%	20%	169
62						–99%	15%	172
63	32					–100%	10%	174
64						–100%	5%	177
65	33	17	9	5		–100%	0%	180
66						–100%	–5%	183
67	34					–100%	–10%	186
68						–99%	–15%	188
69	35	18				–98%	–20%	191
70						–97%	–24%	194
71	36					–96%	–29%	197
72						–94%	–34%	200
73	37	19	10			–92%	–38%	203
74						–90%	–43%	205
75	38					–88%	–47%	208
76						–86%	–51%	211
77	39	20				–83%	–56%	214
78						–80%	–60%	217
79	40					–77%	–63%	219
80						–74%	–67%	222
81	41	21	11	6	3	–71%	–71%	225
82						–67%	–74%	228
83	42					–63%	–77%	231
84						–60%	–80%	233
85	43	22				–56%	–83%	236
86						–51%	–86%	239

Table 2. Relative Current and Step Directions (continued)

1/32 STEP	1/16 STEP	1/8 STEP	1/4 STEP	1/2 STEP	FULL STEP 70%	WINDING CURRENT A	WINDING CURRENT B	ELECTRICAL ANGLE
87	44					–47%	–88%	242
88						–43%	–90%	245
89	45	23	12			–38%	–92%	248
90						–34%	–94%	250
91	46					–29%	–96%	253
92						–24%	–97%	256
93	47	24				–20%	–98%	259
94						–15%	–99%	262
95	48					–10%	–100%	264
96						–5%	–100%	267
97	49	25	13	7		0%	–100%	270
98						5%	–100%	273
99	50					10%	–100%	276
100						15%	–99%	278
101	51	26				20%	–98%	281
102						24%	–97%	284
103	52					29%	–96%	287
104						34%	–94%	290
105	53	27	14			38%	–92%	293
106						43%	–90%	295
107	54					47%	–88%	298
108						51%	–86%	301
109	55	28				56%	–83%	304
110						60%	–80%	307
111	56					63%	–77%	309
112						67%	–74%	312
113	57	29	15	8	4	71%	–71%	315
114						74%	–67%	318
115	58					77%	–63%	321
116						80%	–60%	323
117	59	30				83%	–56%	326
118						86%	–51%	329
119	60					88%	–47%	332
120						90%	–43%	335
121	61	31	16			92%	–38%	338
122						94%	–34%	340
123	62					96%	–29%	343
124						97%	–24%	346
125	63	32				98%	–20%	349
126						99%	–15%	352
127	64					100%	–10%	354
128						100%	–5%	357

8.3.6 nRESET, nENBL, and nSLEEP Operation

The nRESET pin, when driven active low, resets internal logic, and resets the step table to the home position. It also disables the H-bridge drivers. The STEP input is ignored while nRESET is active.

The nENBL pin is used to control the output drivers and enable/disable operation of the indexer. When nENBL is low, the output H-bridges are enabled, and rising edges on the STEP pin are recognized. When nENBL is high, the H-bridges are disabled, the outputs are in a high-impedance state, and the STEP input is ignored.

Driving nSLEEP low will put the device into a low power sleep state. In this state, the H-bridges are disabled, the gate drive charge pump is stopped, the V3P3OUT regulator is disabled, and all internal clocks are stopped. In this state all inputs are ignored until nSLEEP returns inactive high. When returning from sleep mode, some time (approximately 1 ms) needs to pass before applying a STEP input, to allow the internal circuitry to stabilize. Note that nRESET and nENABL have internal pulldown resistors of approximately 100 kΩ. The nSLEEP pin has an internal pulldown resistor of 1 MΩ. nSLEEP and nRESET signals need to be driven to logic high for device operation.

8.3.7 Protection Circuits

The DRV8825 is fully protected against undervoltage, overcurrent, and overtemperature events.

8.3.7.1 Overcurrent Protection (OCP)

An analog current limit circuit on each FET limits the current through the FET by removing the gate drive. If this analog current limit persists for longer than the OCP time, all FETs in the H-bridge will be disabled and the nFAULT pin will be driven low. The device remains disabled until either nRESET pin is applied, or VM is removed and reapplied.

Overcurrent conditions on both high-side and low-side devices; that is, a short to ground, supply, or across the motor winding all result in an overcurrent shutdown. Note that overcurrent protection does not use the current sense circuitry used for PWM current control, and is independent of the I_{SENSE} resistor value or xVREF voltage.

8.3.7.2 Thermal Shutdown (TSD)

If the die temperature exceeds safe limits, all FETs in the H-bridge will be disabled and the nFAULT pin will be driven low. After the die temperature has fallen to a safe level, operation automatically resumes.

8.3.7.3 Undervoltage Lockout (UVLO)

If at any time the voltage on the VM pins falls below the UVLO threshold voltage, all circuitry in the device will be disabled and internal logic will be reset. Operation will resume when $V_{(VMx)}$ rises above the UVLO threshold.

8.4 Device Functional Modes

8.4.1 STEP/DIR Interface

The STEP/DIR interface provides a simple method for advancing through the indexer table. For each rising edge on the STEP pin, the indexer travels to the next state in the table. The direction it moves in the table is determined by the input to the DIR pin. The signals applied to the STEP and DIR pins should not violate the timing diagram specified in Figure 1.

8.4.2 Microstepping

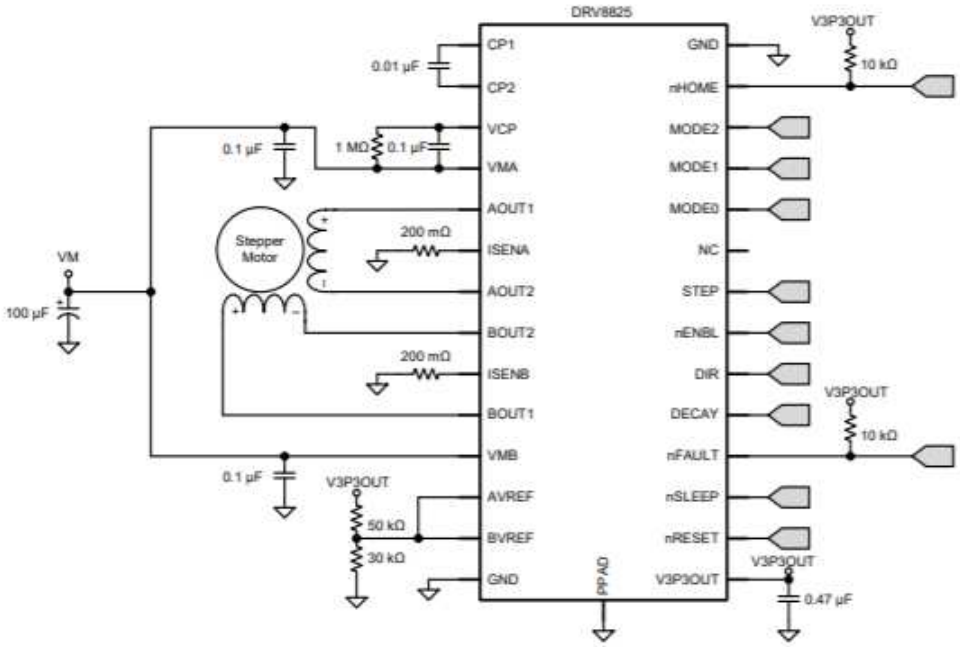
The microstepping indexer allows for a variety of stepping configurations. The state of the indexer is determined by the configuration of the three MODE pins (refer to Table 1 for configuration options). The DRV8825 supports full step up to 1/32 microstepping.

9 Application and Implementation

9.1 Application Information

The DRV8825 is used in bipolar stepper control. The microstepping motor driver provides additional precision and a smooth rotation from the stepper motor. The following design is a common application of the DRV8825.

9.2 Typical Application



9.2.1 Design Requirements

Design Parameter	Reference	Example Value
Supply Voltage	VM	24 V
Motor Winding Resistance	RL	3.9 Ω
Motor Winding Inductance	IL	2.9 mH
Motor Full Step Angle	θstep	1.8°/step
Target Microstepping Level	nm	8 µsteps per step
Target Motor Speed	v	120 rpm
Target Full-Scale Current	IFS	1.25 A

9.2.2 Detailed Design Procedure

9.2.2.1 Stepper Motor Speed

The first step in configuring the DRV8825 requires the desired motor speed and microstepping level. If the target application requires a constant speed, then a square wave with frequency f_{step} must be applied to the STEP pin.

If the target motor startup speed is too high, the motor will not spin. Make sure that the motor can support the target speed or implement an acceleration profile to bring the motor up to speed.

For a desired motor speed (v), microstepping level (n_m), and motor full step angle (θ_{step}).

$$f_{step}\,(\mu steps/second) = \frac{v\left(\frac{rotations}{minute}\right) \times 360\left(\frac{^\circ}{rotation}\right) \times n_m\left(\frac{\mu steps}{step}\right)}{60\left(\frac{seconds}{minute}\right) \times \theta_{step}\left(\frac{^\circ}{step}\right)} \quad (2)$$

$$f_{step}\,(\mu steps/second) = \frac{120\left(\frac{rotations}{minute}\right) \times 360\left(\frac{^\circ}{rotation}\right) \times 8\left(\frac{\mu steps}{step}\right)}{60\left(\frac{seconds}{minute}\right) \times 1.8\left(\frac{^\circ}{step}\right)} \quad (3)$$

θ_{step} can be found in the stepper motor data sheet or written on the motor itself.

For the DRV8825, the microstepping level is set by the MODE pins and can be any of the settings in Table 1. Higher microstepping will mean a smoother motor motion and less audible noise, but will increase switching losses and require a higher f_{step} to achieve the same motor speed.

9.2.2.2 Current Regulation

In a stepper motor, the set full-scale current (I_{FS}) is the maximum current driven through either winding. This quantity depends on the xVREF analog voltage and the sense resistor value (R_{SENSE}). During stepping, I_{FS} defines the current chopping threshold (I_{TRIP}) for the maximum current step. The gain of DRV8825 is set for 5 V/V.

$$I_{FS}\,(A) = \frac{xVREF\,(V)}{A_v \times R_{SENSE}\,(\Omega)} = \frac{xVREF\,(V)}{5 \times R_{SENSE}\,(\Omega)} \quad (4)$$

To achieve I_{FS} = 1.25 A with R_{SENSE} of 0.2 Ω, xVREF should be 1.25 V.

9.2.2.3 Decay Modes

The DRV8825 supports three different decay modes: slow decay, fast decay, and mixed decay. The current through the motor windings is regulated using a fixed-frequency PWM scheme. This means that after any drive phase, when a motor winding current has hit the current chopping threshold (I_{TRIP}), the DRV8825 will place the winding in one of the three decay modes until the PWM cycle has expired. Afterward, a new drive phase starts.

The blanking time, t_{BLANK}, defines the minimum drive time for the current chopping. I_{TRIP} is ignored during t_{BLANK}, so the winding current may overshoot the trip level.

9.2.3 Application Curves

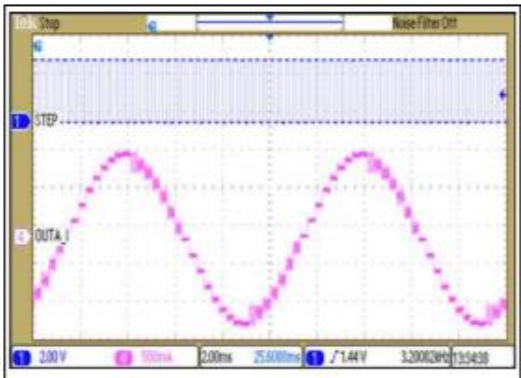


Figure 8. Microstepping Current (Phase A) vs STEP Input, Mixed Decay

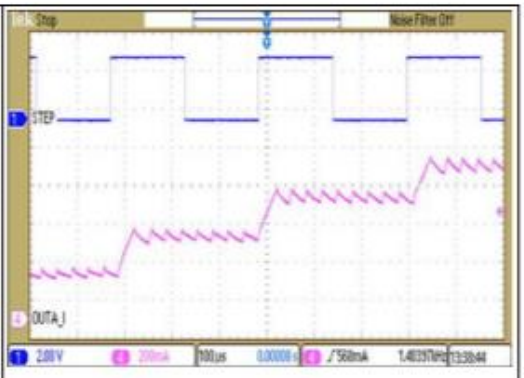


Figure 9. Microstepping Current (Phase A) vs STEP Input, Slow Decay on Increasing Steps

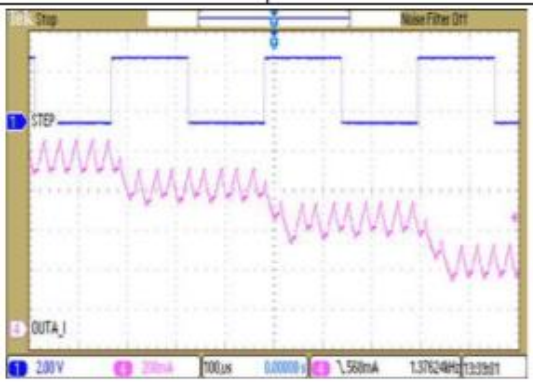


Figure 10. Microstepping Current (Phase A) vs STEP Input, Mixed Decay on Decreasing Steps

10 Power Supply Recommendations

The DRV8825 is designed to operate from an input voltage supply (VMx) range between 8.2 and 45 V. Two 0.1-µF ceramic capacitors rated for VMx must be placed as close as possible to the VMA and VMB pins respectively (one on each pin). In addition to the local decoupling caps, additional bulk capacitance is required and must be sized accordingly to the application requirements.

10.1 Bulk Capacitance

Bulk capacitance sizing is an important factor in motor drive system design. It is dependent on a variety of factors including:

- Type of power supply
- Acceptable supply voltage ripple
- Parasitic inductance in the power supply wiring
- Type of motor (brushed DC, brushless DC, stepper)
- Motor startup current
- Motor braking method

The inductance between the power supply and motor drive system will limit the rate current can change from the power supply. If the local bulk capacitance is too small, the system will respond to excessive current demands or dumps from the motor with a change in voltage. You should size the bulk capacitance to meet acceptable voltage ripple levels.

The data sheet generally provides a recommended value but system level testing is required to determine the appropriate sized bulk capacitor.

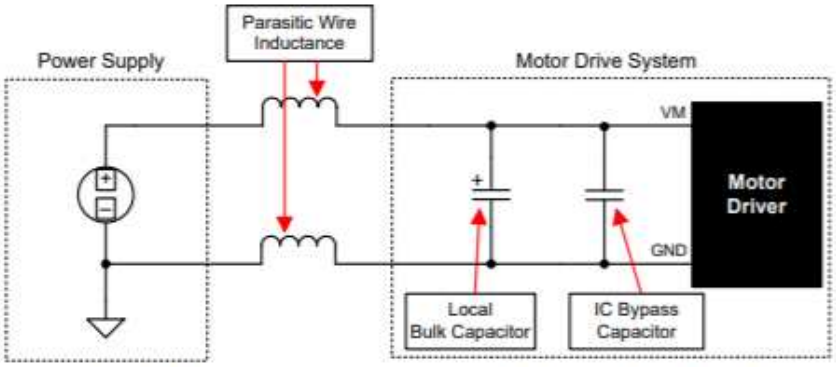


Figure 11. Setup of Motor Drive System With External Power Supply

10.2 Power Supply and Logic Sequencing

There is no specific sequence for powering-up the DRV8825. It is okay for digital input signals to be present before VMx is applied. After VMx is applied to the DRV8825, it begins operation based on the status of the control pins.

11 Layout

11.1 Layout Guidelines

The VMA and VMB pins should be bypassed to GND using low-ESR ceramic bypass capacitors with a recommended value of 0.1-µF rated for VMx. This capacitor should be placed as close to the VMA and VMB pins as possible with a thick trace or ground plane connection to the device GND pin.

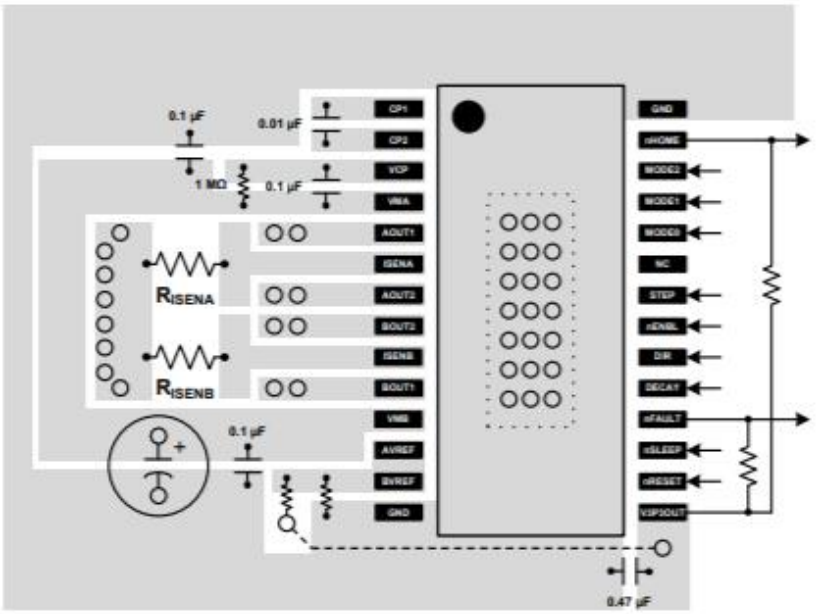
The VMA and VMB pins must be bypassed to ground using an appropriate bulk capacitor. This component may be an electrolytic and should be located close to the DRV8825.

A low-ESR ceramic capacitor must be placed in between the CPL and CPH pins. TI recommends a value of 0.01-µF rated for VMx. Place this component as close to the pins as possible.

A low-ESR ceramic capacitor must be placed in between the VMA and VCP pins. TI recommends a value of 0.1-µF rated for 16 V. Place this component as close to the pins as possible. Also, place a 1-MΩ resistor between VCP and VMA.

Bypass V3P3 to ground with a ceramic capacitor rated 6.3 V. Place this bypass capacitor as close to the pin as possible

11.2 Layout Example



11.3 Thermal Protection

The DRV8825 has thermal shutdown (TSD) as described above. If the die temperature exceeds approximately 150°C, the device will be disabled until the temperature drops to a safe level.

Any tendency of the device to enter TSD is an indication of either excessive power dissipation, insufficient heatsinking, or too high an ambient temperature.

11.3.1 Power Dissipation

Power dissipation in the DRV8825 is dominated by the power dissipated in the output FET resistance, or $R_{DS(ON)}$. Average power dissipation when running a stepper motor can be roughly estimated by Equation 5.

Thermal Protection (continued)

$$P_{TOT} = 4 \times R_{DS(ON)} \times \left(I_{OUT(RMS)}\right)^2 \quad (5)$$

where P_{TOT} is the total power dissipation, $R_{DS(ON)}$ is the resistance of each FET, and $I_{OUT(RMS)}$ is the RMS output current being applied to each winding. $I_{OUT(RMS)}$ is equal to the approximately 0.7x the full-scale output current setting. The factor of 4 comes from the fact that there are two motor windings, and at any instant two FETs are conducting winding current for each winding (one high-side and one low-side).

The maximum amount of power that can be dissipated in the device is dependent on ambient temperature and heatsinking.

Note that $R_{DS(ON)}$ increases with temperature, so as the device heats, the power dissipation increases. This must be taken into consideration when sizing the heatsink.

11.3.2 Heatsinking

The PowerPAD™ package uses an exposed pad to remove heat from the device. For proper operation, this pad must be thermally connected to copper on the PCB to dissipate heat. On a multi-layer PCB with a ground plane, this can be accomplished by adding a number of vias to connect the thermal pad to the ground plane. On PCBs without internal planes, copper area can be added on either side of the PCB to dissipate heat. If the copper area is on the opposite side of the PCB from the device, thermal vias are used to transfer the heat between top and bottom layers.

For details about how to design the PCB, refer to TI application report SLMA002, "PowerPAD™ Thermally Enhanced Package" and TI application brief SLMA004, *PowerPAD™ Made Easy*, available at www.ti.com.

In general, the more copper area that can be provided, the more power can be dissipated. It can be seen that the heatsink effectiveness increases rapidly to about 20 cm^2, then levels off somewhat for larger areas.

12 Device and Documentation Support

12.1 Trademarks

PowerPAD is a trademark of Texas Instruments.

12.2 Electrostatic Discharge Caution

These devices have limited built-in ESD protection. The leads should be shorted together or the device placed in conductive foam during storage or handling to prevent electrostatic damage to the MOS gates.

12.3 Glossary

SLYZ022 — *TI Glossary*.

This glossary lists and explains terms, acronyms, and definitions.

13 Mechanical, Packaging, and Orderable Information

The following pages include mechanical, packaging, and orderable information. This information is the most current data available for the designated devices. This data is subject to change without notice and revision of this document. For browser-based versions of this data sheet, refer to the left-hand navigation.

4.- ASDC Series Silicon Pressure Sensors[4]

ASDX Series Silicon Pressure Sensors

Low Pressure and Ultra-Low Pressure Digital Output, ±2% Total Error Band, 10 Inches H_2O to 100 psi

DESCRIPTION

The ASDX Series is a Silicon Pressure Sensor offering either an I^2C or SPI digital interface for reading pressure over the specified full scale pressure span and temperature range.

The ASDX is fully calibrated and temperature compensated for sensor offset, sensitivity, temperature effects and non-linearity using an on-board Application Specific Integrated Circuit (ASIC). Calibrated output values for pressure are updated at approximately 1 kHz.

The standard ASDX is calibrated over the temperature range of 0 °C to 85 °C [32 °F to 185 °F]. The sensor is characterized for operation from a single power supply of either 3.3 Vdc or 5.0 Vdc.

These sensors are available to measure absolute, differential and gage pressures. The absolute versions have an internal vacuum reference and an output value proportional to absolute pressure. Differential versions allow application of pressure to either side of the sensing diaphragm. Gage versions are referenced to atmospheric pressure and provide an output proportional to pressure variations from atmosphere.

The ASDX Series sensors are intended for use with non-corrosive, non-ionic working fluids such as air and dry gases. They are designed and manufactured according to standards in ISO 9001.

FEATURES

- Output options: I^2C- or SPI-compatible 12-bit digital
- Precision ASIC conditioning and temperature compensated over 0 °C to 85 °C [32 °F to 185 °F] temperature range
- Low operating voltage
- Absolute, differential and gage types
- Pressure ranges from 10 inches H_2O to 100 psi
- Standard calibrations in inches H_2O, cm H_20, psi, mbar, bar, kPa
- Total error band of ±2.0% of full scale span maximum
- RoHS compliant

POTENTIAL APPLICATIONS

- Flow calibrators
- Ventilation and air flow monitors
- Gas flow instrumentation
- Sleep apnea monitoring and therapy equipment
- Barometry
- Pneumatic controls
- HVAC

[4] https://sensing.honeywell.com/honeywell-sensing-asdx-series-digital-pressure-sensors-product-sheet-008095-13-en.pdf

ASDX Series Silicon Pressure Sensors

Table 1. Absolute Maximum Ratings[1]

Parameter	Min	Max	Unit
Supply voltage (V_{supply})	-0.3	6.6	Vdc
Voltage to any pin	-0.5	V_{supply} + 0.3	Vdc
Digital clock frequency: I²C SPI	 100 50	 400 800	kHz
ESD susceptibility (human body model)	3	-	kV
Storage temperature	-50 [-58]	125 [257]	°C [°F]
Lead temperature (2 s to 4 s)	-	250 [482]	°C [°F]
External capacitance between V_{supply} and ground[2]	100	470	nF

Table 2. Operating Specifications

Parameter	Min.	Typ.	Max.	Unit
Supply voltage (V_{supply})[3] 3.3 Vdc 5.0 Vdc Sensors are either 3.3 Vdc or 5.0 Vdc per the Order Guide (see Figure 1)	 3.0 4.75	 3.3[3] 5.0[3]	 3.6 5.25	Vdc
Supply current	2.0	3.5	5.0	mA
Compensated temperature range[4]	0 [32]	-	85 [185]	°C [°F]
Operating temperature range[5]	-20 [-4]	-	105 [221]	°C [°F]
Overpressure[6]	2X operating pressure range minimum			
Burst pressure[7]	3X operating pressure range minimum			
Startup time (power up to data ready)	-	2.8	7.5	ms
Response time	-	0.46	-	ms
I²C or SPI voltage level low	-	-	0.2	V_{supply}
I²C or SPI voltage level high	0.8	-	-	V_{supply}
Pull-up on SDA and SCL (I²C output only)	1	-	-	kOhm
Total error band[8]	-	-	2.0	%FSS[9][10]
Output resolution	12	-	-	bits

Table 3. Environmental Specifications

Parameter	Characteristic
Humidity	0% to 95% RH non-condensing
Vibration	10 G at 20 Hz to 2000 Hz
Shock	100 G for 11 ms
Life	1 million cycles minimum

Table 4. Wetted Materials[11]

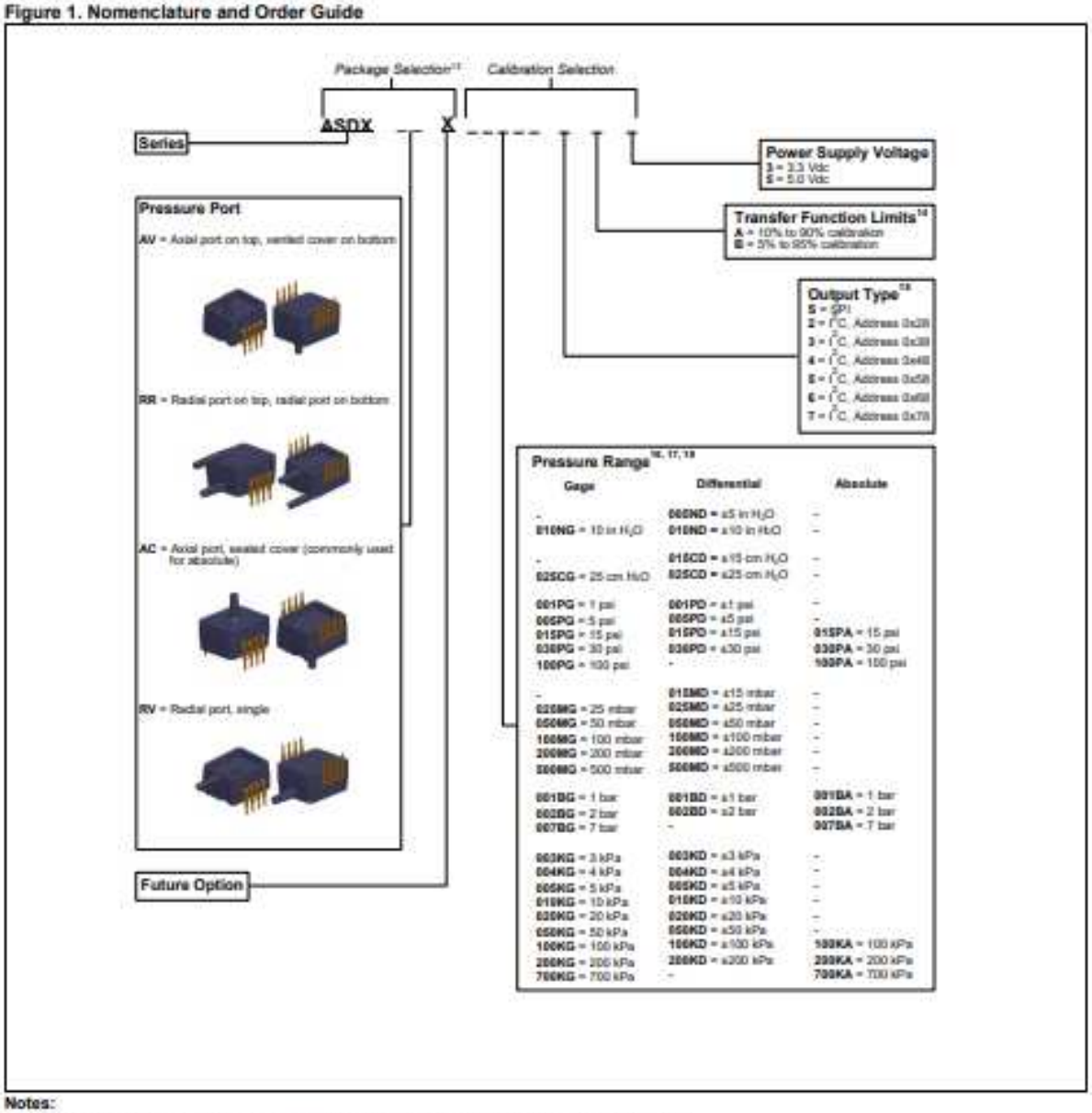
Parameter	Port 1 (Pressure Port)[12]	Port 2 (Reference Port)[12]
Covers	glass-filled PBT	glass-filled PBT
Adhesives	silicone	silicone and epoxy
Electronic components	silicon and glass	silicon, glass, and gold

Notes:

1. Absolute maximum ratings are the extreme limits that the device will withstand without damage to the device.
2. An external bypass capacitor is **required** across the supply voltage (Pins 6 and 3 – see Figure 4) as close to the sensor supply pin as possible for correct sensor operation.
3. Ratiometricity of the sensor (the ability of the output to scale to the input voltage) is achieved within the specified operating voltage for each option. Other custom supply voltages are available, please contact Honeywell Customer Service.
4. The sensor is not reverse polarity protected. Incorrect application of excitation voltage or ground to the wrong pin may cause electrical failure.
5. The compensated temperature range is the temperature range (or ranges) over which the sensor will produce an output proportional to pressure within the specified performance limits.
6. The operating temperature range is the temperature range over which the sensor will produce an output proportional to pressure but may not remain within the specified performance limits.
7. Overpressure is the maximum pressure which may safely be applied to the product for it to remain in specification once pressure is returned to the operating pressure range. Exposure to higher pressures may cause permanent damage to the product.
8. Burst pressure is the maximum pressure that may be applied to any port of the product without causing escape of pressure media. Product should not be expected to function after exposure to any pressure beyond the burst pressure.
9. Total error band is the maximum deviation in output from ideal transfer function over the entire compensated temperature and pressure range. Includes all errors due to offset, full scale span, pressure non-linearity, pressure hysteresis, repeatability, thermal effect on offset, thermal effect on span and thermal hysteresis. Specification units are in percent of full scale span (%FSS).
10. Full scale span (FSS) is the algebraic difference between the output signal measured at the maximum (Pmax.) and minimum (Pmin.) limits of the pressure range.
11. Consult Honeywell Customer Service for detailed material information.
12. For AC pressure port configuration, the "pressure" and "reference" ports are reversed.

Low and Ultra-Low Pressure Digital Output

Figure 1. Nomenclature and Order Guide



Notes:

13. Other package combinations are possible, please contact Honeywell Customer Service.
14. The transfer function limits define the output of the sensor at a given pressure input. By specifying the output signal at the maximum (Pmax.) and minimum (Pmin.) limits of the pressure range, the complete transfer curve for the sensor is defined. See Figure 2 for a graphical representation of each calibration. For the 12-bit digital output, Table 6 provides the output of the sensor at significant percentages. These outputs are valid at the rated input voltage of the sensor.
15. The output type defines which communication protocol the sensor uses to communicate. Available protocols are I^2C or half duplex SPI (sensor acts only as a slave). This communication protocol is not field selectable, and must be defined when ordering the sensor.
16. Custom pressure ranges are available, please contact Honeywell Customer Service.
17. The pressure units (inches H_2O, cm H_2O, psi, mbar, bar, kPa) define the units used during calibration and in the application.
18. See Table 5 for an explanation of sensor types.

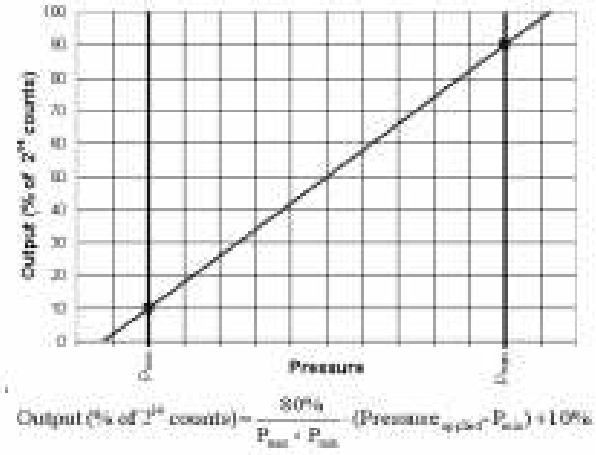
ASDX Series Silicon Pressure Sensors

Table 5. Sensor Types

Type	Description
Absolute	Output is proportional to difference between applied pressure and built-in reference to vacuum (zero pressure).
Gage	Output is proportional to difference between applied pressure and atmospheric (ambient) pressure.
Differential	Output is proportional to difference between pressure applied to each of the pressure ports (Port 1 – Port 2).

Figure 2. Transfer Functions and Limits

A Calibration, 10% to 90%



B Calibration, 5% to 95%

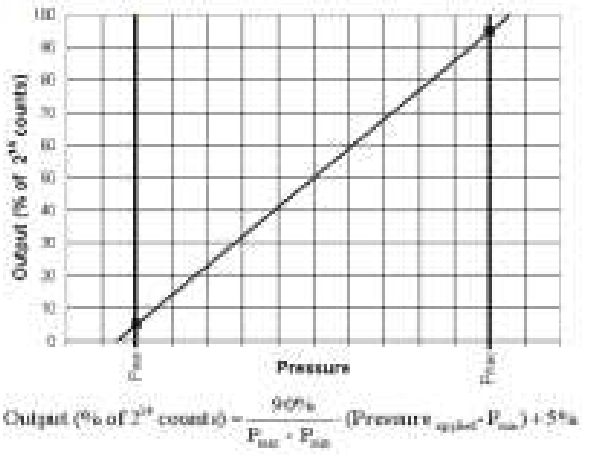
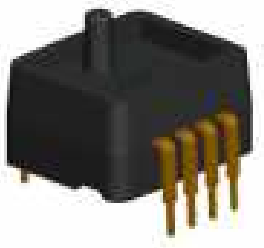


Table 6. Sensor Output at Significant Percentages

% Output	Digital Counts (dec)	Digital Counts (hex)
0%	0	0x0000
5%	819	0x0333
10%	1639	0x0666
50%	8192	0x2000
90%	14746	0x399A
95%	15565	0x3CCD
100%	16383	0x3FFF

Figure 3. Completed Catalog Listing Example

ASDXAVX001PG2A3:
AV pressure port, 1 psi gage, I²C output (Address 0x28), 10% to 90% calibration at 3.3 Vdc operation.



ASDXAVX001PG2A3 Output vs Pressure

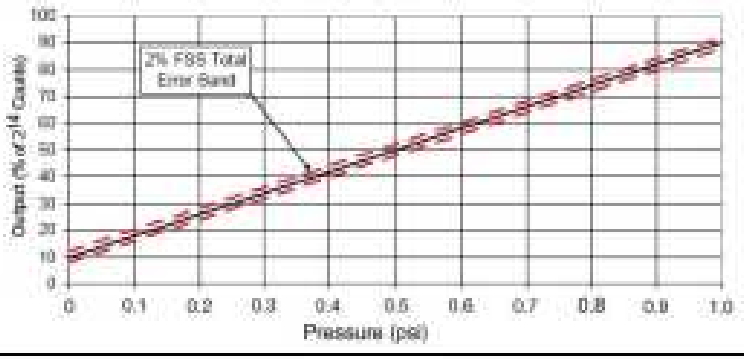


Figure 4. Dimensional Drawings (For reference only: mm [in].)

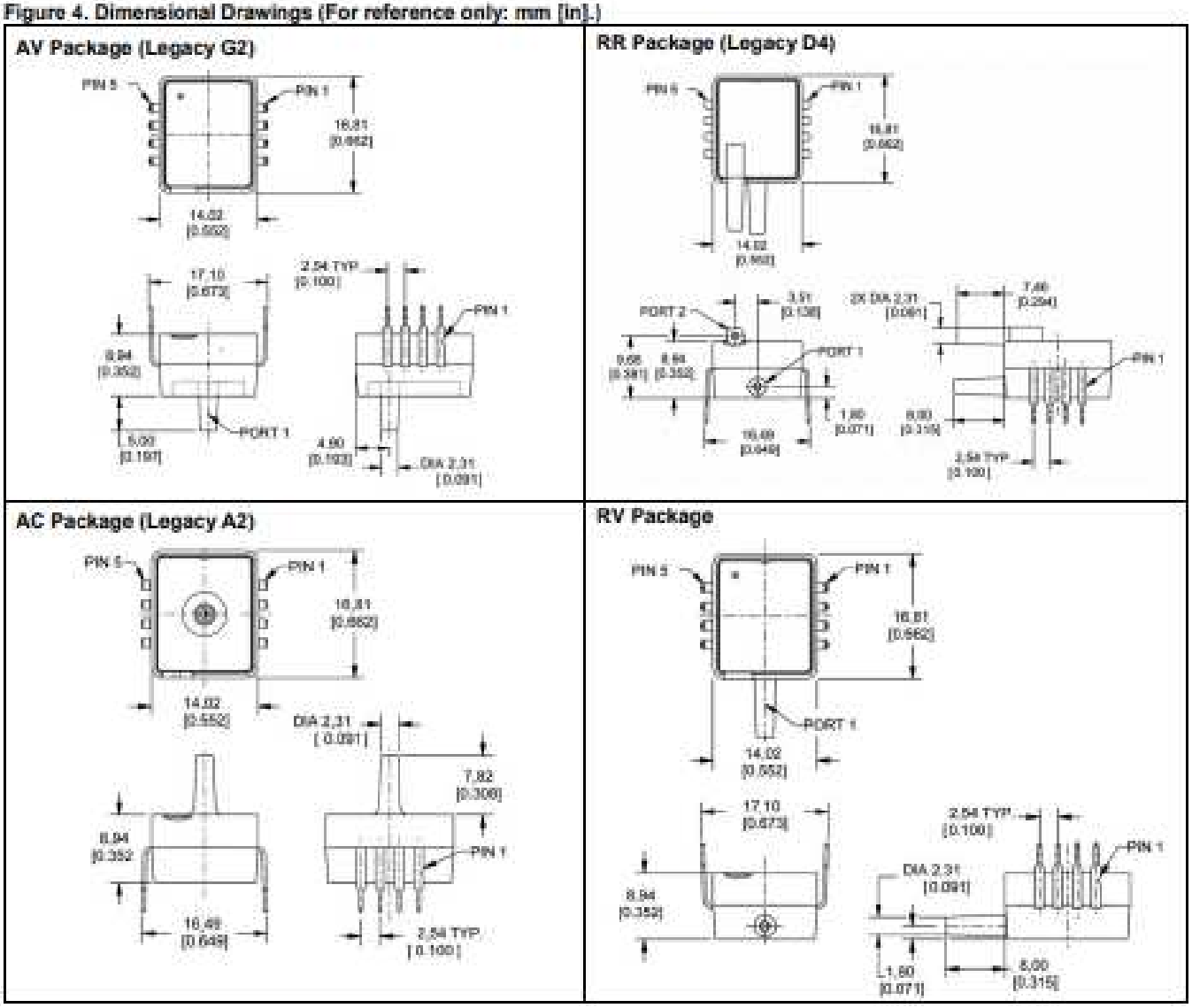


Table 7. Pinout

I²C				SPI			
Pin	Definition	Type	Description	Pin	Definition	Type	Description
1	SDA	digital I/O	serial bidirectional data; data is clocked in or out on clock edge of SCL	1	MISO	digital output	"Master In Slave Out" - serial output data; data is clocked out on clock edge of SCK
2	SCL	digital input	serial clock input; used to clock data on SDA	2	SCK	digital input	serial clock input; used to clock data on MISO
3	GND	supply	power supply ground	3	GND	supply	power supply ground
4	N/C	not used	do not connect in the application	4	N/C	not used	do not connect in the application
5	SS	digital output	interrupt signal (conversion complete output)	5	SS	digital input	slave select
6	Vsupply	supply	power supply source	6	Vsupply	supply	power supply source
7	N/C	not used	do not connect in the application	7	N/C	not used	do not connect in the application
8	N/C	not used	do not connect in the application	8	N/C	not used	do not connect in the application

⚠ WARNING

PERSONAL INJURY

DO NOT USE these products as safety or emergency stop devices or in any other application where failure of the product could result in personal injury.

Failure to comply with these instructions could result in death or serious injury.

WARRANTY/REMEDY

Honeywell warrants goods of its manufacture as being free of defective materials and faulty workmanship. Honeywell's standard product warranty applies unless agreed to otherwise by Honeywell in writing; please refer to your order acknowledgement or consult your local sales office for specific warranty details. If warranted goods are returned to Honeywell during the period of coverage, Honeywell will repair or replace, at its option, without charge those items it finds defective. **The foregoing is buyer's sole remedy and is in lieu of all other warranties, expressed or implied, including those of merchantability and fitness for a particular purpose. In no event shall Honeywell be liable for consequential, special, or indirect damages.**

While we provide application assistance personally, through our literature and the Honeywell web site, it is up to the customer to determine the suitability of the product in the application.

Specifications may change without notice. The information we supply is believed to be accurate and reliable as of this printing. However, we assume no responsibility for its use.

⚠ WARNING

MISUSE OF DOCUMENTATION

- The information presented in this product sheet is for reference only. Do not use this document as a product installation guide.
- Complete installation, operation, and maintenance information is provided in the instructions supplied with each product.

Failure to comply with these instructions could result in death or serious injury.

SALES AND SERVICE

Honeywell serves its customers through a worldwide network of sales offices, representatives and distributors. For application assistance, current specifications, pricing or name of the nearest Authorized Distributor, contact your local sales office or:

E-mail: info.sc@honeywell.com

Internet: sensing.honeywell.com

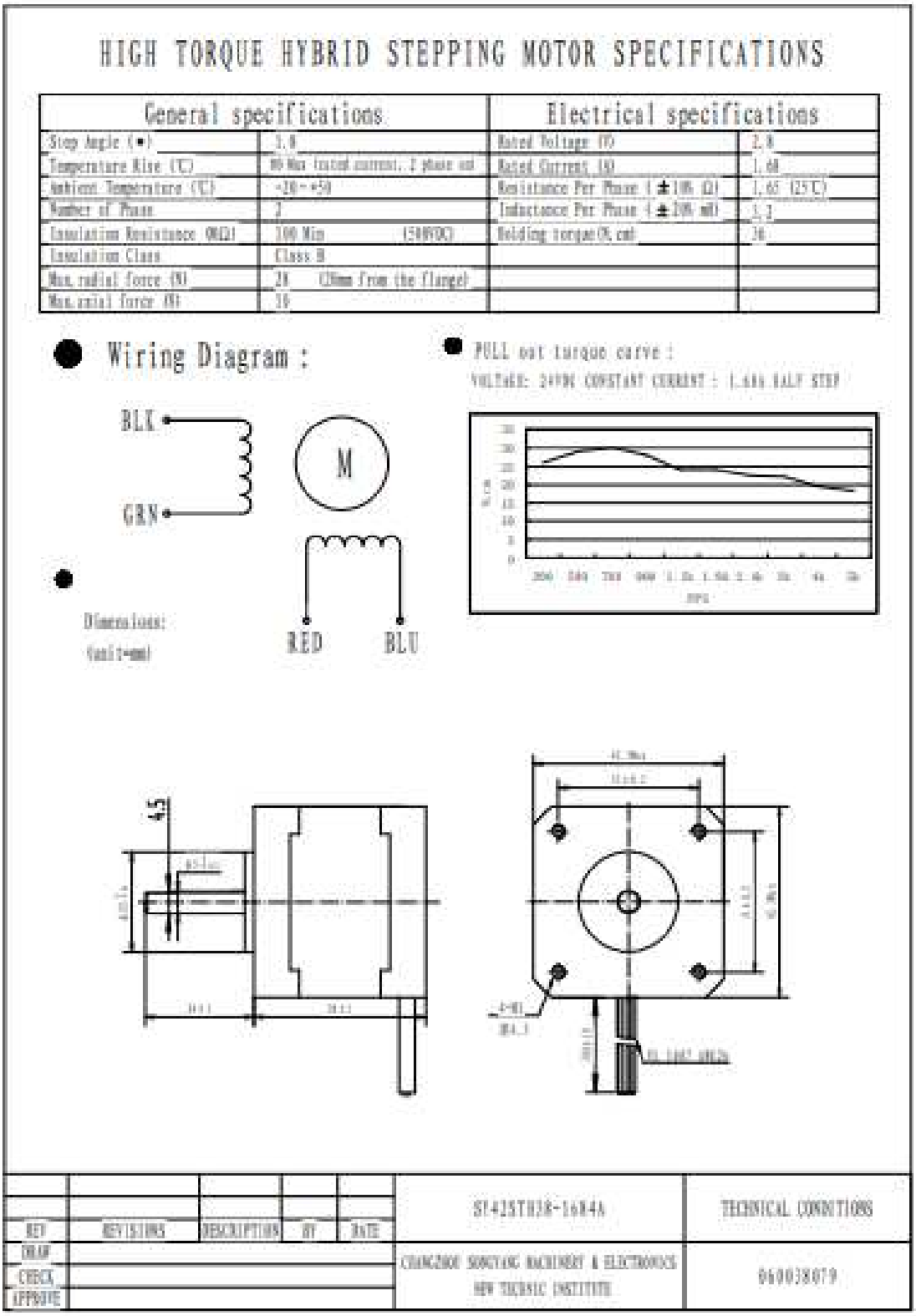
Phone and Fax:

Asia Pacific	+65 6355-2828; +65 6445-3033 Fax
Europe	+44 (0) 1698 481481; +44 (0) 1698 481676 Fax
Latin America	+1-305-805-8188; +1-305-883-8257 Fax
USA/Canada	+1-800-537-6945; +1-815-235-6847 +1-815-235-6545 Fax

Sensing and Control
Honeywell
1985 Douglas Drive North

Honeywell

5.- NEMA 17 Stepper Motor Datasheet[5]



HIGH TORQUE HYBRID STEPPING MOTOR SPECIFICATIONS

General specifications		Electrical specifications	
Step Angle (°)	1.8	Rated Voltage (V)	2.8
Temperature Rise (℃)	80 Max (rated current, 2 phase on)	Rated Current (A)	1.68
Ambient Temperature (℃)	-20~+50	Resistance Per Phase (±10% Ω)	1.65 (25℃)
Number of Phase	2	Inductance Per Phase (±20% mH)	3.2
Insulation Resistance (MΩ)	100 Min (500VDC)	Holding torque(N.cm)	36
Insulation Class	Class B		
Max.radial force (N)	28 (20mm from the flange)		
Max.axial force (N)	10		

● Wiring Diagram :

● PULL out torque curve :

VOLTAGE: 24VDC CONSTANT CURRENT : 1.68A HALF STEP

● Dimensions:

(unit:mm)

REV	REVISIONS	DESCRIPTION	BY	DATE	ST42STH28-1684A	TECHNICAL CONDITIONS
DRAW					CHANGZHOU SONGYANG MACHINERY & ELECTRONICS NEW TECHNIC INSTITUTE	060038079
CHECK						
APPROVE						

6.- Código Completo

6.1.- Main Code

```
// ****************** Libraries
#include <Arduino.h>
#include <TimerOne.h> // No analogWrite() in pins 9 y 10
#include <Wire.h>
#include <ADXL345V.h>
#include <EEPROMex.h>
#include <PID_v1.h>
#include <RTClib.h>
#include <SoftwareSerial.h>
#include <EasyTransferI2C.h>

// ****************** Modules
#include "control.h"
#include "data.h"
#include "process.h"
#include "user.h"
#include "communication.h"
#include "sensors.h"
#include "DHT.h"

#define TICK_MS 10
#define CYCLE_MS 50
#define MAX_CONT_TEST 20 // 6000 =  5 minuto

unsigned long ContTest = 0;


void setup()
{
  startProcess();
  startUser();
  startCommunication();
  startData();
  startControl();

  ContTest = 0;
/*
  Timer1.initialize(TICK_MS * 1000); // initialize timer1, and set
0,1s
*/
  Timer1.attachInterrupt(isr_Timer1); // attaches isr overflow
interrupt
}

void isr_Timer1()
{
        tickProcess();
        tickData();
        tickControl();
        tickUser();
        tickCommunication();
```

[5] https://www.pololu.com/file/0J714/SY42STH38-1684A.pdf

```
}

void FullTest()
{
        testProcess();
        testData();
        testControl();
        testUser();
        testCommunication();
}

char end_condition() { return 0;}

void FullEnd() {
    endProcess();
    endUser();
    endCommunication();
    endData();
    endControl();
}

void loop()
{
        cycleProcess();
        cycleData();
        cycleControl();
        cycleUser();
        cycleCommunication();

        if (ContTest++ > MAX_CONT_TEST) {
                ContTest = 0;
                FullTest();
        }

        if (end_condition()) FullEnd();

        delay (CYCLE_MS);
}
```

6.2.- Control Module

```
// CHANGES MADE IN PRINT FUNCTIONS IN ORDER TO WORK WITH ANDROID APP
DEVELOPED
//***************LIBRARIES & MODULES********************
#include <Arduino.h>
#include <Wire.h>
#include <ADXL345V.h>
#include <PID_v1.h>
#include <SoftwareSerial.h>
#include <EasyTransferI2C.h>

#include "RTClib.h"
#include "RTC.h"
#include "control.h"
#include "filter.h"
#include "science.h"
#include "record.h"
```

```
#include "BasicStepperDriver.h"
#include "SDCard.h"
//*******************************************************

#define miSerial Serial // OJO poner Serial3 para blue (Bluetooth now
connected to Serial Port 0, same as USB)
#define miSerialEvent serialEvent

//I2C communication for TFT screen & CTD Sensors
EasyTransferI2C UNO;
EasyTransferI2C CTD;
#define MASTER_ADDR 7  //I2C Master
#define UNO_ADDRESS 8 //I2C slave address
#define CTD_ADDRESS 9  //I2C CTD sensors slave address


struct CTD_DATA{
  int16_t depth;
  float tempA;
  float tempB;
  float tempC;
  float conduct;
};

CTD_DATA ctd_data;

struct SEND_DATA{
    int depth;
    int temp;
    double inAngle;
    int posRail;
    double setAngle;
    int margin;
    int hours;
    int minutes;
    int seconds;
};

SEND_DATA science;

//Struc to save the values coming from sensor in different modules
struct sciData{
  int depthH;
  int temp;
};


#define SW1 6
#define SW2 7

#define MAX_RAIL 120  //Length of the endless screw after testing with
some margin (MAX = 275)
#define MOTOR_STEPS 200 //200 steps per revolution (1.8°)
#define MICROSTEPS 4 //1/4 steps (Best method for smoothness and
torque)

//Pin configuration for motor
#define DIR 10 //(9)
#define STEP 11 //(8)
#define ENABLE 13
```

```
int stepRPM = 120; //Speed of Stepper Motor
int RPM_a = 0;
long delta_tetha=0; //Angle difference for P control
unsigned long timer=0, delta_t=0; //delta time or how long it takes to
execute data acquisition

//Variables for measurement of On/Off motor periods
unsigned long TOnFWAcc = 0; //Forward tray motion
unsigned long TOnBWAcc = 0; //Backwards tray motion
unsigned long Mstart = 0;  //Measure of the cycle period

int accx,accy,accz;    // integer Read from Accel

double setAngle, inAngle, outAngle; //Define Variables we'll be
connecting to in PID
double angleOFF=0; //Angle offset for trimming procedure
//double kp = 2.0, ki = 0.0, kd = 0.0;
int margin = 10;
int numCycle=0, showAccel=0;
int posRail=60;
unsigned long timerChange = 0;

//SD information log variables.
int m=0, s=0, M_OnFW=0, M_OnBW=0;
unsigned long MesMillis = 0;


BasicStepperDriver stepper(MOTOR_STEPS, DIR, STEP, ENABLE);  //Stepper
driver declaration

// PID::PID(double* inAngle, double* outAngle, double* setAngle,
double Kp, double Ki, double Kd, int ControllerDirection)
//PID myPID(&inAngle, &outAngle, &setAngle, kp, ki, kd, DIRECT);

ADXL345 adxl; //variable adxl is an instance of the ADXL345 library

//SoftwareSerial softSerial(31, 30); // RX, TX for Bluetooth

//CTD sensor variable for reception
int16_t depth=0;
float tempA=0, tempB=0, tempC=0, conduct=0;

//************** Functions
*****************************************************************

void command(const char* cmd, int num_bytes_response) {
        delay(1000);
        //miSerial.print(cmd);
        delay(1500);
        for (int i=0;i<num_bytes_response;i++)
        Serial.write(miSerial.read());
}

void initBlue() {
        command("AT",2);// response: OK
        command("AT+VERSION",12);// response: OKlinvorV1.5
        command("AT+NAMEGlider3",9);//response: OKsetname
        command("AT+BAUD4",8);//response: OK9600
        command("AT+PIN1234",1);//response:
}
```

```
void readSensors()
{
}

void writeActuators()
{
}

void receiveEvent(int numBytes){
  }

void startControl()
{
  //Communication
        miSerial.begin(9600);
  Wire.begin(MASTER_ADDR);
  UNO.begin(details(science), &Wire);
  CTD.begin(details(ctd_data), &Wire);
  Wire.onReceive(receiveEvent);

  //RTC module
  //initializeRtc(); //Use only once, the battery will keep the time

        // Blue
        //initBlue();  //Uncomment only for configuration of new BT
module

        // Accelerometer
        adxl.powerOn();

  //SD Card Initialization
  SDSetup();
  CreateMotorLogFile();
  CreateSensorLogFile();
  CreateHoneyWellLogFile();

  // Timer functions
  //RTC data adquisition
  m = getMinute();
  s = getSecond();
  Mstart = millis(); //Starting value of millis when RTC is measured
  timer = millis();  // For Filter
  WriteMotorValues(m, s, Mstart, M_OnFW, M_OnBW);  //Log into SD card
  WriteSensorValues(m, s, Mstart, depth, tempA, tempB, tempC,
conduct); //Log sensors values to SD

        // Stepper Motor
  stepper.begin(stepRPM, MICROSTEPS);  //Motor initial parameters
  stepper.enable();

        // SW
        pinMode(SW1, INPUT_PULLUP);
        pinMode(SW2, INPUT_PULLUP);

  setAngle = 0;  //Angle initialization

        /* PID
  inAngle = 0.0;
  myPID.SetMode(AUTOMATIC);
  myPID.SetOutputLimits(40, 200);
  */
```

```
  //miSerial.println("startControl");
}

void cycleControl()
{
  numCycle++;
        readSensors();
        writeActuators();
  M_OnFW = 0;
  M_OnBW = 0;

//Check if data is received from the CTD sensors and assign values to
local variables
  if(CTD.receiveData()){
    depth = ctd_data.depth;
    tempA = ctd_data.tempA;
    tempB = ctd_data.tempB;
    tempC = ctd_data.tempC;
    conduct = ctd_data.conduct;
    MesMillis = millis();
    WriteSensorValues(m, s, MesMillis, depth, tempA, tempB, tempC,
conduct);
  }

  //Send Real time to screen (Too time consuming, other method will be
assessed)
  /*
  science.hours = getHour();
  science.minutes = getMinute();
  science.seconds = getSecond();
  UNO.sendData(UNO_ADDRESS);
  */

  //Set parameter to be sent to screen
  science.setAngle = setAngle;
  science.margin = margin;
  science.depth = depth;
  science.temp = tempA;

  stepper.disable(); //Endless screw does not slip. No motor brake
needed.


        adxl.readAccel(&accx, &accy, &accz); //read the accelerometer
values and store them in variables  x,y,z
        delta_t = millis() - timer;                          //
calculate time through loop i.e. acq. rate
        timer = millis();                                      // reset
timer

        inAngle = kalmanCalculate(accx, 0.0, delta_t);
  inAngle = (inAngle - angleOFF);  //Compensate observed offset in
accel

  //P speed controller
  delta_tetha = (setAngle - inAngle);
  delta_tetha = abs(delta_tetha);

  RPM_a = delta_tetha*0.90; //In order to extend the range between min
and max.
```

```
  if (RPM_a >= 200){
    stepRPM = 200;
  }else if (RPM_a <= 60){   //Limiting the minimum speed to ensure
smooth motion.
    stepRPM = 60;
  } else stepRPM = RPM_a;

  //myPID.Compute();
  stepper.begin(stepRPM, MICROSTEPS);
  stepper.disable();

/*
        if (digitalRead(SW1)==0) {
                stepper.stop();
                posRail=MAX_RAIL+1;
        }
        if (digitalRead(SW2)==0) {
                stepper.stop();
                posRail=0;
        }
*/
        if (posRail<0){
          posRail=0;
          science.posRail = posRail;
        }
        if (posRail>MAX_RAIL){
          posRail=MAX_RAIL+1;
          science.posRail = posRail;
        }

                if (inAngle > setAngle + margin) {
                        if (posRail<MAX_RAIL) {
              stepper.enable();
              delay(20);

              //SD card Log
              MesMillis = millis();
              M_OnFW = 1;
              WriteMotorValues(m, s, MesMillis, M_OnFW, M_OnBW);

              stepper.move(-MICROSTEPS*92); //the motor moves the tray
exactly 1mm back or forth

              MesMillis = millis();
              M_OnFW = 0;
              WriteMotorValues(m, s, MesMillis, M_OnFW, M_OnBW);

              posRail++;
              science.posRail = posRail;
              science.inAngle = inAngle;
                              miSerial.print(inAngle);
                              miSerial.print(" ");
                              miSerial.print(posRail);
                              miSerial.print(" ");
              miSerial.print(depth);
              UNO.sendData(UNO_ADDRESS);
      }
                }
                else if (inAngle < setAngle - margin) {
                        if (posRail>0) {
            stepper.enable();
```

```
              delay(20); //Delay introduced to avoid power bank
automatic shutdown

              MesMillis = millis();
              M_OnBW = -1;
              WriteMotorValues(m, s, MesMillis, M_OnFW, M_OnBW);

              stepper.move(MICROSTEPS*92);

              MesMillis = millis();
              M_OnBW = 0;
              WriteMotorValues(m, s, MesMillis, M_OnFW, M_OnBW);

              posRail--;
              science.posRail = posRail;
              science.inAngle = inAngle;
                              miSerial.print(inAngle);
              miSerial.print(" ");
              miSerial.print(posRail);
              miSerial.print(" ");
              miSerial.print(depth);
              UNO.sendData(UNO_ADDRESS);
       }
                }

        if (numCycle > timerChange && timerChange>1000) {
                  numCycle=0;
                  setAngle = -1 * setAngle;
                  //miSerial.print(" changed angle: ");
                  //miSerial.println(setAngle);
        }

  if (showAccel) {
   // outAngle x,y,z values - Commented out
    miSerial.println(accx);
    miSerial.print(", ");
    miSerial.print(inAngle);
    miSerial.print(", ");
    //  miSerial.print(y);
    //  miSerial.print(", ");
    //  miSerial.println(z);
    miSerial.println(stepRPM);
    //miSerial.print(" Period On FW: ");
    //miSerial.println(TOnFW);
    //miSerial.print(" Period On BW: ");
    //miSerial.println(TOnBW);
    //miSerial.print(" Cycle Period: ");
    //miSerial.println(Ttotal);
    }
  else {
    if ((numCycle%20)==0) {
      m = getMinute();
      s = getSecond();
      //miSerial.print(inAngle);
      //miSerial.print(" ");
    }
  }
  MesMillis = millis(); //Starting value of millis when RTC is
measured
  WriteMotorValues(m, s, MesMillis, M_OnFW, M_OnBW);  //Log into SD
card
```

```
}


void miSerialEvent()
{
  while (miSerial.available()){
        char ch;
        ch = (char)miSerial.read();
        //miSerial.print("Leido ");
        //miSerial.println(ch);
        switch (ch) {
                  case '1':
                stepper.enable();
                delay(20);
                            stepper.move(MICROSTEPS*910);
                posRail = posRail - 10;
                            break;
                 case '2':
                stepper.enable();
                delay(20);
                            stepper.move(-MICROSTEPS*910);
                posRail = posRail + 10;
                            break;
      case '3':
                stepper.enable();
                delay(20);
                stepper.move(MICROSTEPS*450);
                posRail = posRail - 5;
                break;
      case '4':
                stepper.enable();
                delay(20);
                stepper.move(-MICROSTEPS*450);
                posRail = posRail + 5;
                break;
      case '5':
                stepper.enable();
                delay(20);
                stepper.move(MICROSTEPS*240);
                posRail = posRail - 3;
                break;
      case '6':
                stepper.enable();
                delay(20);
                stepper.move(-MICROSTEPS*240);
                posRail = posRail + 3;
                break;
      case 'S':posRail = 60; break; //Once the load is trimmed, set
current position as middle position
                  case 's':stepper.stop();
               angleOFF = inAngle;
               setAngle = 0;
                            break;
                  case 'I':stepRPM++; break;
                  case 'i':stepRPM--; break;
                  case 'A':setAngle++;
               //miSerial.println(setAngle);
               //miSerial.print(" ");
               //miSerial.print(margin);
                            break;
```

```
                  case 'a':setAngle--;
                //miSerial.println(setAngle);
                //miSerial.print(" ");
                //miSerial.print(margin);
                            break;
                  case 'M':margin++;
                //miSerial.println(setAngle);
                //miSerial.print(" ");
                //miSerial.print(margin);
                            break;
                  case 'm':margin--;
                //miSerial.println(setAngle);
                //miSerial.print(" ");
                //miSerial.print(margin);
                            break;
        case 'T':setAngle = 25;
                //miSerial.println(setAngle);
                //miSerial.print(" ");
                //miSerial.print(margin);
                break;
        case 't':setAngle = -25;
                //miSerial.println(setAngle);
                //miSerial.print(" ");
                //miSerial.print(margin);
                break;
        case 'H':angleOFF = inAngle; break;
        case 'h':setAngle = 0; break;
        case 'V':showAccel = 1; break;
        case 'v':showAccel = 0; break;
                  case 'X':timerChange+=1200; numCycle=0; break;
                  case 'x':timerChange-=1200; numCycle=0; break;
        case 'Q':storeYes=1; numCycle=0; break;
                  case 'q':storeYes=0; numCycle=0; break;
        case 'W':showrecordAngle();break;
            break;
          }
          //miSerial.print(" SP MotorSpeed: ");
          //miSerial.println(stepRPM);
          //miSerial.print(" A angle: ");
          //miSerial.println(setAngle);
          //miSerial.print(" M margin: ");
          //miSerial.println(margin);
          //miSerial.print(" X change: ");
          //miSerial.println(timerChange);
          //miSerial.print(" Q store: ");
          //miSerial.println(storeYes);

          //miSerial.println(" T Angle 10, H Horizontal, s Stop, M
margin, V vervose");
    }
}

void tickControl()
{
}

void testControl()
{
  struct sciData theSci;
        theSci = cycleScience(inAngle, tempA);
  //Honeywell dataLOG
```

```
  MesMillis = millis();
  WriteHoneyValues(m, s, MesMillis, theSci.depthH);  //Log into SD
card

  science.depth = depth;
  science.temp = tempA;
  science.inAngle = inAngle;
  science.posRail = posRail;
  UNO.sendData(UNO_ADDRESS);

  miSerial.print(inAngle);
  miSerial.print(" ");
  miSerial.print(posRail);
  miSerial.print(" ");
  miSerial.print(depth);
}

void endControl()
{
}
```

6.3.- SD Card Module

```
#include <SD.h>
#include <Arduino.h>

//SPI settings
//MOSI,MISO,SCLK set as default
int CS_pin = 53;
int SDin = 1;

void SDSetup(){

 Serial.println("Initializing Card");
  pinMode(CS_pin, OUTPUT);

  //Check if card is ready
  if(!SD.begin(CS_pin)){
    Serial.println("Card Failed!!");
    SDin=0;
    return;
  }
  Serial.println("Card Ready");
  SDin=1;
}

void CreateMotorLogFile(){
  File logFile = SD.open("LogSTEPL.csv", FILE_WRITE); //Movement of
the longitudinal tray
  if (logFile){
    logFile.println(", , , ,"); //Blank line
    String header =
"RTC_Min,   RTC_sec,   Millis,   Motor_FW,   Motor_BW";
    logFile.println(header);
    logFile.close();
  }else if (SDin==1){
    Serial.println("Couldn't open log file");
  }
}
```

```
void CreateSensorLogFile(){
  File logFile = SD.open("LogCTD.csv", FILE_WRITE); //Movement of the
longitudinal tray
  if (logFile){
    logFile.println(", , , , , , ,"); //Blank line
    String header =
"RTC_Min,   RTC_sec,   Millis,   Depth,   TemperatureA,   TemperatureB
,   TemperatureC,   Conductivity";
    logFile.println(header);
    logFile.close();
  }else if (SDin==1){
    Serial.println("Couldn't open log file");
  }
}

void CreateHoneyWellLogFile(){
  File logFile = SD.open("LogHONEY.csv", FILE_WRITE); //Movement of
the longitudinal tray
  if (logFile){
    logFile.println(", , , "); //Blank line
    String header = "RTC_Min,   RTC_sec,   Millis,   Depth";
    logFile.println(header);
    logFile.close();
  }else if (SDin==1){
    Serial.println("Couldn't open log file");
  }
}

void WriteMotorValues (int RTC_mins, int RTC_sec, unsigned long
MesMillis, int motor_FW, int motor_BW){
  //CSV format data string
  String dataString = String(RTC_mins)+ ",    " + String(RTC_sec) +
",    " + String(MesMillis) + ",    " + String(motor_FW) + ",    " +
String(motor_BW);

  //Open file to write to, only one file open at a time
  File logFile = SD.open("LogSTEPL.csv", FILE_WRITE);
  if(logFile){
    logFile.println(dataString);
    logFile.close();
  }else if (SDin==1){
    Serial.println("Couldn't access file");
  }
}

void WriteSensorValues (int RTC_mins, int RTC_sec, unsigned long
MesMillis, int16_t depth, float tempA, float tempB, float tempC, float
conduc){
  //CSV format data string
  String dataString = String(RTC_mins)+ ",    " + String(RTC_sec) +
",    " + String(MesMillis) + ",    " + String(depth) + ",    " +
String(tempA)+ ",    " + String(tempB) + ",    " + String(tempC)+
",    " + String(conduc);

  //Open file to write to, only one file open at a time
  File logFile = SD.open("LogCTD.csv", FILE_WRITE);
  if(logFile){
    logFile.println(dataString);
    logFile.close();
  }else if (SDin==1){
```

```
    Serial.println("Couldn't access file");
  }
}

void WriteHoneyValues (int RTC_mins, int RTC_sec, unsigned long
MesMillis, int depth){
  //CSV format data string
  String dataString = String(RTC_mins)+ ",     " + String(RTC_sec) +
",     " + String(MesMillis) + ",     " + String(depth);

  //Open file to write to, only one file open at a time
  File logFile = SD.open("LogHONEY.csv", FILE_WRITE);
  if(logFile){
    logFile.println(dataString);
    logFile.close();
  }else if (SDin==1){
    Serial.println("Couldn't access file");
  }
}
```

6.4.- Bluetooth Configuration Module

```
#include <Arduino.h>
#include "bluetoothSetup.h"

char name[10] = "GLIDER";
char bps = '4';
char password[10] = "1234";

void configureBluetooth()
{
        pinMode(12,OUTPUT); //Changed to pin 12 as pin 13 is used by
the motor
        digitalWrite(12,HIGH);
        //If bluetooth is not conected
        delay(10000);
        digitalWrite(12,LOW);

        //Start configuration
        Serial.print("AT");
        delay(1000);

        //Configure name
        Serial.print("AT+NAME");
        Serial.print(name);
        delay(1000);

        //configure baud
        Serial.print("AT+BAUD");
        Serial.print(bps);
        delay(1000);

        //Configure password
        Serial.print("AT+PIN");
        Serial.print(password);
        delay(1000);
}
```

6.5.- Kalman Filter Module

```
/*

Extended Kalman filter applied to one axis of the accelerometer

*/

#include <math.h>
#include <MatrixMath.h>
#define PI 3.14159265358979f

//************************ Filter Variables
**********************************
float x_angle=0;

float Q_angle  =  0.01; //0.001    //0.005
float Q_gyro   =  0.0;  //0.003  //0.0003
float R_angle  =  0.01;  //0.03     //0.008

float x_bias = 0;
float P_00 = 0, P_01 = 0, P_10 = 0, P_11 = 0;
float y, S;
float K_0, K_1;

float kalmanCalculate(float newAngle, float newRate, int looptime)
{
    float dt = float(looptime)/1000;
    x_angle += dt * (newRate - x_bias);
    P_00 +=  - dt * (P_10 + P_01) + Q_angle * dt;
    P_01 +=  - dt * P_11;
    P_10 +=  - dt * P_11;
    P_11 +=  + Q_gyro * dt;

    y = newAngle - x_angle;
    S = P_00 + R_angle;
    K_0 = P_00 / S;
    K_1 = P_10 / S;

    x_angle +=  K_0 * y;
    x_bias  +=  K_1 * y;
    P_00 -= K_0 * P_00;
    P_01 -= K_0 * P_01;
    P_10 -= K_1 * P_00;
    P_11 -= K_1 * P_01;

    return x_angle;
}
```

6.6.- RTD Module

```
#include "rtc.h"
#include <Wire.h>
#include "RTClib.h"

RTC_DS3231 RTC;
DateTime now;
```

```
//Initializes the rtc time to the current time of the PC
void initializeRtc()
{
        //Wire.begin(); // Start the I2C port
        RTC.begin(); // Initiates communication with the RTC
        //RTC.adjust(DateTime(__DATE__, __TIME__)); // Sets the date
and time of compilation
}

int getYear()
{
        now = RTC.now();
        return now.year();
}

int getMonth()
{
        now = RTC.now();
        return now.month();
}

int getDay()
{
        now = RTC.now();
        return now.day();
}

int getHour()
{
        now = RTC.now();
        return now.hour();
}

int getMinute()
{
        now = RTC.now();
        return now.minute();
}

int getSecond()
{
        now = RTC.now();
        return now.second();
}
```

6.7.- Science Module

```
#include <Arduino.h>
#include "science.h"
#include "communication.h"
#include "record.h"
#include "sensors.h"
#include <Wire.h>
```

```
struct sciData{
  int depth;
  int temp;
};

//objects to store the information from the sensors
Record recordAngle(1024,2047);
Record recordDepth(2048,3071);
Record recordTemp(3072,4096);

int contador =0;
byte storeYes=0;

void startScience()
{

}

void showrecordAngle()
{
   recordAngle.loadEE();
   recordDepth.loadEE();
   recordTemp.loadEE();
}

struct sciData cycleScience(double angle, float tempA)
{
  struct sciData sci;

        int  depthH;

        //angle = mission.getAngle();


        depthH = (int)(getPressure()); //Si queremos quedarnos con un
decimal
  sci.depth = depthH;

        //temp = getTemperatureZX();
  sci.temp = tempA;

        sendUnder(angle, depthH, tempA);

         //showVar("A ", angle);
         //showVar("D ", depthH);
         //showVar("T ", tempA);
         //blueSerial.println();
         //sendUnder((int)angle, depth, temperature);

/*
         if (contador++ > 10 && storeYes) {
             recordAngle.storeEE((char)angle);
             recordDepth.storeEE((char)depth);
             recordTempB.storeEE((char)temp);
             contador=0;
             blueSerial.print("!");
         }
*/
 return sci;
}
```

6.8.- Sensors Module

```
/*


SENSOR PRESION
modelo ASDX RRX 100PG 2A5
RR: Radial Radial
X: nada
100PG: 100PSI
2: I2C ADDR 0X28
A: RANGE 10% A 90%
5: VCC = 5V


#include "sensors.h"
#include <Wire.h>
#include "DHT.h"

#define DHTPIN 2               //Select pin2 to comunicate
#define DHTTYPE DHT11  //The DHT11 is selected

#define I2C_PRESSION 40        //Unique bus address for pressure sensor
#define MEGA
#define EXT_REF 0

#define ADC_HONEYWELL 2 // Arduino analog input pin
#define ADC_ZXTEMP       1 // Arduino analog input pin

const float Null = 0.50; // Null VDC; datasheet Page 32
const float Sensitivity = 266.6;  // Sensitivity mV/psi; datasheet
Page 32
// A variable that will be used by Arduino to communicate with the
sensor starts
DHT dht(DHTPIN, DHTTYPE);

//Pressure sensor
byte msb, lsb = 0;
int press = 0;
int out_Max = 14745;
int out_Min = 1600;
int P_max = 90; //psi (max.: 6.12 atm)
int P_min = 15; //psi (max.: 1.02 atm)
int P_out = 0;

//Moisture sensor
byte relePin = 8;     //pin del rele
byte saPin = 13;      //pin del la salida analogica (led 13 con PWM)
byte shPin = A1;      //Pin del sensor de humedad
int valHumedad = 0;     // valor de la humedad
byte valSalida = 0;     // valor de la humedad en byte

//sensors function to initialize
void startSensors()
{
        //Start pressure sensor, wakes up I2C bus
        Wire.begin();
        //Start temperature and humidity sensor
```

```
        dht.begin();
        pinMode(relePin,OUTPUT); // Fran: ¿para que es esto?
}

void getdata(byte *a, byte *b)
{
  Wire.requestFrom(I2C_PRESSION,2);    //Sends content of first two
registers
  while (Wire.available()){  //Salve may send less than requested
  *a = Wire.read();                    //first byte recieved stored
here
  *b = Wire.read();                    //second byte recieved stored
here
  }
}

//The pressure is read
float getPressure()
{
  getdata(&msb,&lsb);
  //Serial.print("byte 1: "); Serial.println(aa,BIN);
  //Serial.print("byte 2 "); Serial.println(bb,BIN);
  press = msb;
  press = (press << 8) + lsb;
  //Serial.print("Combined byte: "); Serial.println(c,BIN);
  //Serial.print("Count #: "); Serial.println(c);
  P_out = (((press - out_Min)*(P_max - P_min))/(out_Max - out_Min)) +
P_min; //Conversion found in datasheet
  //Serial.print(" Pressure: "); Serial.print(pressure,DEC);
//Serial.println(" psi");

  return P_out;
}

float getPressure2() {
  float pressurePSI,pressureMBAR,pressureVDC;
  int pressure;

  pressure = analogRead(ADC_HONEYWELL);
  pressureVDC = (float)pressure * 0.0048828125; // (5/1024 =
0.0048828125)
  pressureVDC = pressureVDC - Null;
  pressurePSI = pressureVDC / Sensitivity * 1000;
  pressureMBAR = pressurePSI * 68.948;

  return pressureMBAR;
}

//function that returns the temperature sensor DTH11
int getTemperature()
{
  return dht.readTemperature();
}

//function that returns the humidity sensor DTH11
int getHumidity()
{
  return dht.readHumidity();
}
```

```
//function that returns the humidity read by the YL-69 sensor. Returns
255 if no humidity and low humidity value if it detects
int getMoisture()
{
        valHumedad = analogRead(shPin); // reads the value of the
moisture
        valSalida = map(valHumedad, 0, 1023, 0, 255); // sets the value
at analog output range
        analogWrite(saPin,valSalida);
        //Serial.print("Humedad: ");
        //Serial.println(valSalida);

        return valSalida;
}


// Read from ZX-Thermometer (Analog with long wire plus  thermistor)
// Return temp x10:  234 for 23.4°C
int getTemperatureZX()
{
        int sensorValue;
    double temp;
        sensorValue = analogRead(ADC_ZXTEMP);
        temp = (sensorValue - 151) / 10.5;
         return(temp*10);
}
```