

Control and Instrumentation of a Super-Maneuverable Unmanned Underwater Vehicle

MASTER THESIS

In partial fulfillment of the requirements for the degree

„Master of Science in Engineering“

Master program:

„Mechatronics & Smart Technologies“

Management Center Innsbruck

Supervisor:

University of California, Berkeley

Mohammad-Reza Alam, PhD

Internal supervisor:

Bernhard Hollaus, MSc

Author:

Stefan Messner, BSc

1410620030

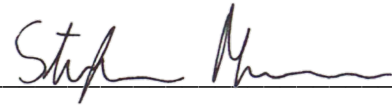
Declaration in Lieu of Oath

I hereby declare, under oath, that this master thesis has been my independent work and has not been aided with any prohibited means. I declare, to the best of my knowledge and belief, that all passages taken from published and unpublished sources or documents have been reproduced whether as original, slightly changed or in thought, have been mentioned as such at the corresponding places of the thesis, by citation, where the extent of the original quotes is indicated.

The paper has not been submitted for evaluation to another examination authority or has been published in this form or another.

Innsbruck, 30.09.2016

Place, Date



Signature

Acknowledgement

A special thank you goes to the Marshall Plan Scholarship Foundation, which financially supported me during my stay at the University of California, Berkeley. Additionally, I want to thank Professor Mohammad-Reza Alam, Mohsen Saadat and Mir Abbas Jalali, my supervisor at Cal, for their support and the hours spent contributing to the project. Moreover, my family and friends for their support and patience.

Abstract

The Super-Maneuverable Unmanned Underwater Vehicle (SUUV) project is developed in cooperation with the University of California, Berkeley under the supervision of Prof. Mohammad-Reza Alam. The work deals with the development of a super-agile autonomous swimmer which is capable of performing full 3D reorientation and translational maneuvers.

The reorientation of the swimmer is realized without using thrusters, wings, rudders, stabilizers or fins and, thus is much more energy efficient and super-agile in comparison to conventional swimmer. Therefore, the swimmer enables a better exploration of the physics, chemistry, and biology of the ocean environment. Another field lies in the inspection of offshore platforms or ship hulls. A Double Gimbal Control Moment Gyro (DGCMG) is used to orientate the swimmer in three-dimensional space (3D). The double gimbal system consists of a high-speed wheel which rotates with up to 12,000 *RPM*. The center wheel creates an inertia platform whereby the double gimbal system enables to rotate the swimmer around it in 3D space. Two counter-rotating thrusters generate enough thrust to propel the robot to perform translational movement. The problem is to develop a novel control method for the orientation control of the Autonomous Undersea Vehicles (AUVs). Fundamental is the deployment of the electronics, the embedded system and the implementation of the computational-expensive control algorithms. The final goal of the project is to develop the instrumentation of the swimming robot, to implement different control methods and to evaluate them regarding stability and controllability. The best controller has to be found and applied. This results in the best maneuverability of the swimmer. In the field of control theory, several methods for stabilizing non-linear dynamic systems are used as Backstepping and Lyapunov control. The expected outcome is to discover findings concerning the best solution for non-linear dynamic control problems and to adapt them on an innovative method for the reorientation of AUVs which can be pioneering for future tasks of the same kind.

The paper shows that the novel developed method using a DGCMG system for the reorientation of underwater vehicles yields excellent increases in the AUVs agility. Further improvements regarding implementation of the final non-linear control algorithms have to be done and the final underwater test has to be performed.

Keywords: Underwater Vehicle, Super-Agile Swimmer, Instrumentation, Non-Linear Control;

Contents

1.	Introduction	1
1.1	State of the Art	3
1.2	Applications	4
2.	Design and Modeling	5
2.1	Swimmers Design.....	5
2.2	Dynamic Model.....	7
2.2.1	Reference Frames	7
2.2.2	Fundamental Equations.....	8
2.2.3	Coordinate System Rotations	10
2.2.4	Assumptions.....	11
2.2.5	Governing Equations	12
3.	Non-Linear Controller Design.....	13
3.1	Control Theory.....	13
3.1.1	Lyapunov Control	13
3.1.2	Backstepping.....	14
3.2	Adaptive Controller Design	15
4.	Electro-Mechanical Construction.....	18
4.1	Embedded System	18
4.1.1	Arduino Genuino Uno Board.....	18
4.1.2	Arduino Mega 2560 Board.....	19
4.1.3	Arduino Due Board	19
4.2	Arduino Open-Source Software (IDE).....	21
4.2.1	Object-Oriented Programming	21
4.2.2	Interrupt Service Routines	22
4.3	Stepper Motors.....	23
4.3.1	Adafruit Motor Shield 2	23
4.3.2	Stepper Motor Driver	24
4.3.3	Driver Input Signal Generation.....	28
4.4	Optical Encoders	31
4.4.1	Interrupt Encoder Read	31
4.4.2	Robogaia Encoder Counter Shield.....	33
4.5	Brushless DC Motor – Gyro.....	35
4.6	Brushless DC Motor – Thrusters.....	37
4.7	Inertial Measurement Unit.....	38
4.8	Arduino / Arduino – Communication	40
4.8.1	Arduino Task Distribution.....	41
4.8.2	Software Implementation	41

4.9	MATLAB / Arduino – Communication	44
4.9.1	Serial Communication.....	44
4.9.2	Ethernet Communication	45
4.10	Software Task Distribution.....	51
4.11	Electrical Circuits.....	53
4.11.1	Electrical Power Supply	54
4.11.2	Logic Signals Connections	54
5.	Manufacturing	56
5.1	Fabrication and Design.....	56
5.2	Final Assembly	56
5.3	Waterproofness	59
6.	Swimmer Agility Tests	60
6.1	One Axis Agility Results.....	60
6.1.1	Hanging Method for One DOF	60
6.1.2	Open-Loop Control.....	61
6.1.3	Close-Loop Control – Yaw Angle.....	62
6.2	Two Axis Agility Results.....	64
6.2.1	Hanging Method for Two DOF	64
6.2.2	Close-Loop Result – Pitch Angle	64
6.3	Results Evaluation.....	65
7.	Conclusions.....	66
7.1	Closure.....	66
7.2	Future Work.....	66
	Bibliography	VII
	List of Figures	X
	List of Listings.....	XII
	List of Abbreviations.....	XIII
	List of Symbols	XIV
A.	Appendix A	XV
B.	Appendix B	XIX
C.	Appendix C	XXII

1. Introduction

Autonomous Undersea Vehicles (AUVs) mostly are orientated in space by thruster, wings, rudders, stabilizer or fins. Such actuation methods are energy consuming and limit the operation range of AUVs. Additionally, orientation methods like these are slow or limited in maneuverability. That means not all maneuvers can be performed with each method. Furthermore, external thrusters for the reorientation generate additional flows in the surrounding fluid. Imagine a method, which allows to perform full 3D on-point reorientation, fast translational maneuvers, and which is much more energy efficient as conventional methods. Let us introducing the Super-Maneuverable Unmanned Underwater Vehicle (SUUV) [1], a project of the University of California, Berkeley.

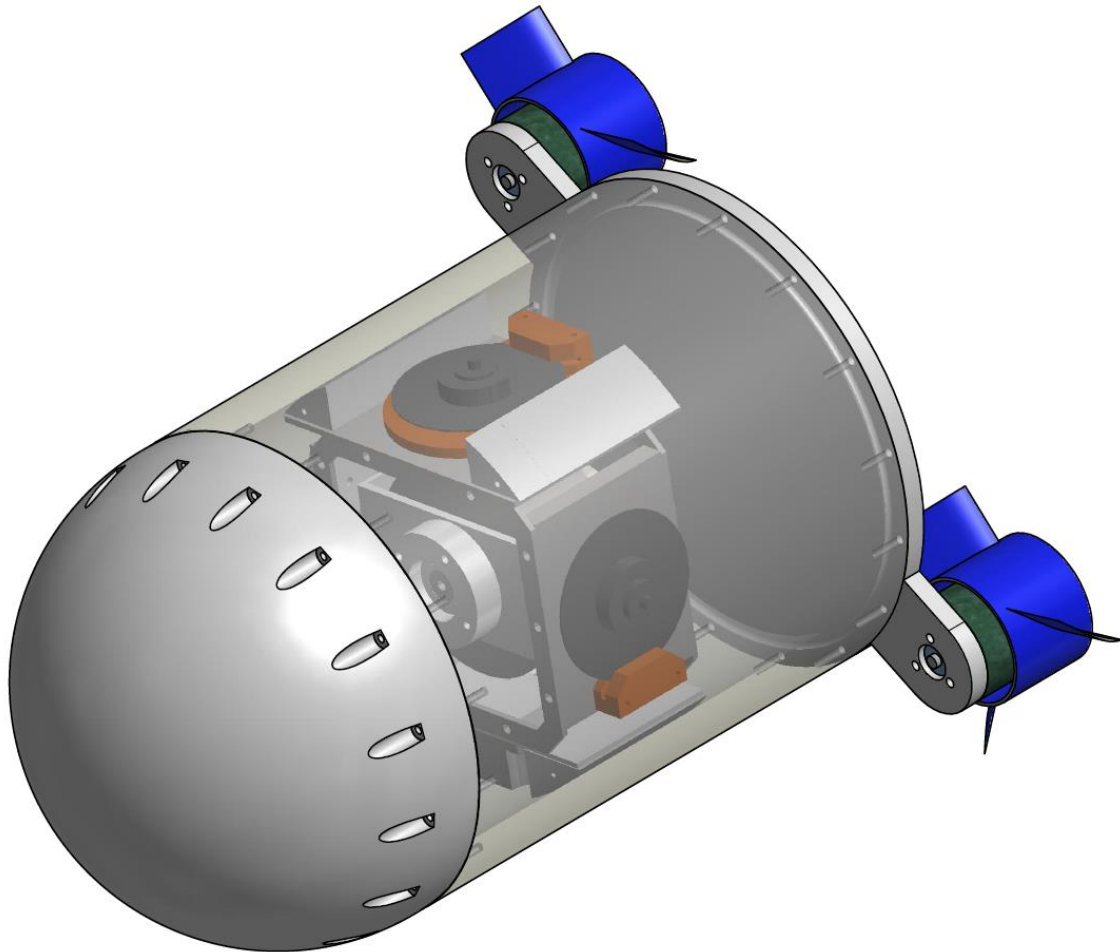


Figure 1-1: Super-Maneuverable Unmanned Underwater Vehicle (SUUV)

A team around Prof. Mohammad-Reza Alam at the Theoretical & Applied Fluid Dynamics Laboratory (TAF Lab) [2] develops a super-agile swimmer. The swimmer has no external wings, fins, rudders or stabilizer which are used for reorientation maneuvers. Therefore, the swimmer is not generating external flows during reorientation. The swimmer is reoriented by using a Double Gimbal Control Moment Gyro (DGCMG). Two counter

rotating thrusters are used to generate a force for the forward and backward propulsion. The DGCMG consists of two gimbals and a high-speed wheel. The wheel spins with up to 12,000 *RPM* and generates an inertia platform which is used to rotate the swimmer. The two gimbals allow to rotate the robot around the gyro in 3D space. Therefore, no external flows are generated during reorientation maneuvers by the SUUV and the swimmer is super agile. Because of its natural buoyancy the swimmer can hold its heave position naturally.

The DGCMG system is first introduced in [3] where it is used to reorientate a satellite in space. Obviously, the dynamics of the system is very different regarding the missing gravity and because no external torques are acting to the body in space. The goal of this thesis is to adapt the dynamics introduced in the previous work, the assembly, the development of the electronics for the swimmer and most importantly prove that the system is working for an undersea robot.

1.1 State of the Art

Control moment gyro systems are nowadays used to reorientate satellites in space [4]. Different systems therefore are developed. Mostly, three spinning wheels are used, whereby the spinning speed of the wheels create a reaction force which reorientate the satellite in space. Such systems can also be used to reorientate objects on earth, whereby the dynamics is completely different because of earth's gravity. Such a system is shown with the "Cubli"-project [5] where three spinning wheels are used to balance a cube. The cube balancing on its edge can be seen in the figure below.

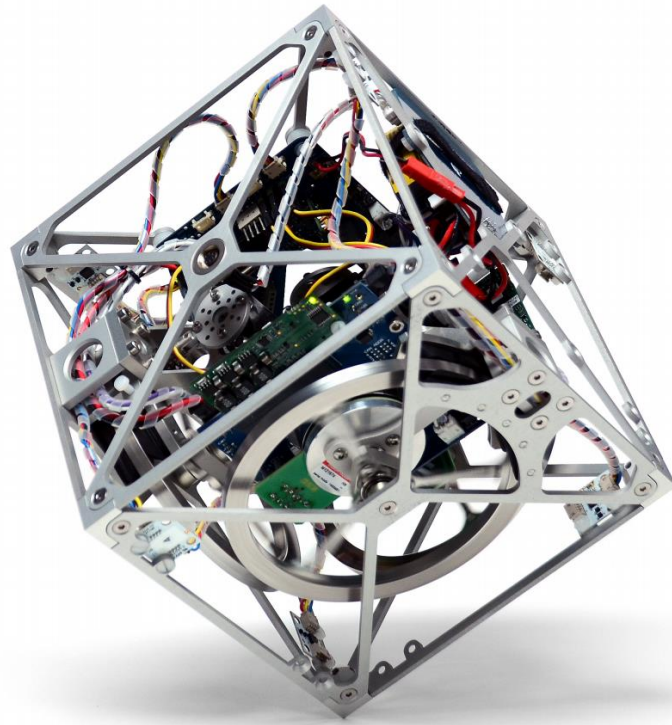


Figure 1-2: "Cubli" – ETH Zurich Project

The cube is able to flip, move and balance on its edge and the corners just by using the reaction forces generated by the three spinning wheels. The marine applications of control moment gyros are given for the stabilization of the ships [6]. Such systems can either be actively or passively controlled, for example in order to reduce vessel motions. The main goal of these systems is to increase the comfort for passengers or to enable a stable working platform.

1.2 Applications

The applications of super-agile swimmers lie in the exploration of sensitive underwater regions. The low additional flows produced by the swimmer and the high maneuverability enable a better exploration of the physics, chemistry, and biology of the environment. [7] Another field lies in the inspection of offshore platforms or ship hulls. The SUUV profits from the high maneuverability at high and low velocities and therefore opens new fields of applications for AUVs. Further, the robot can be build much more compact then conventional robots. As a result of the energy efficacy of the robot, the underwater time for the robot can be increased significantly. This is an essential key for the usability and a long observation time of the underwater environment. The high movability further enables new possibilities regarding swimming in formations of AUVs. [8] Such swimming in formations can enable the creation of an optical communication, as seen below.

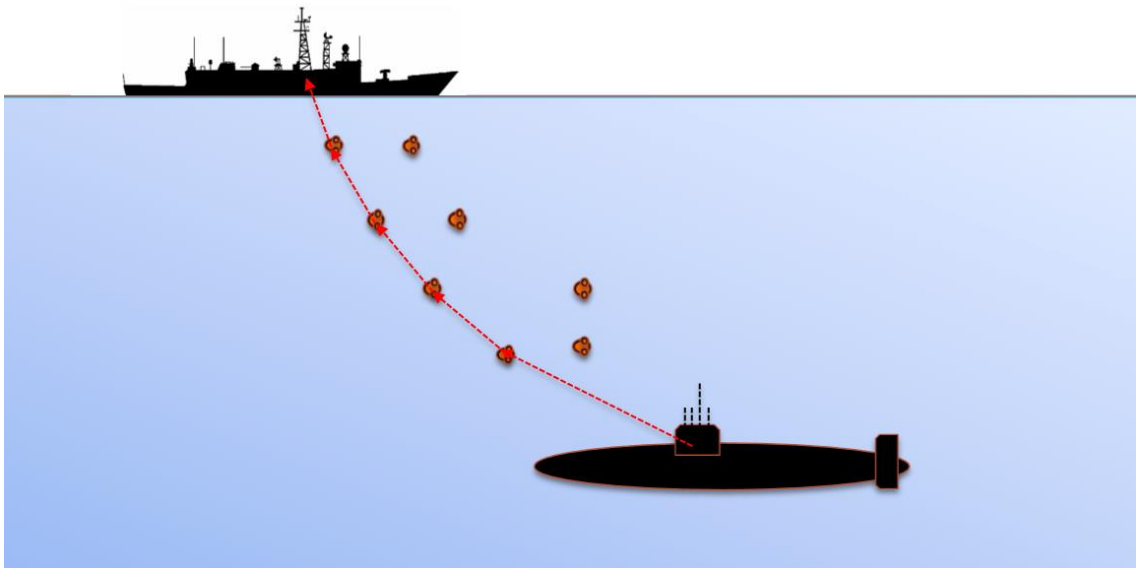


Figure 1-3: Super-Agile Swimmer – Optical Communication Network with Submarine

A group of super-agile swimmers can swim in formation with sight contact and enable the creation of an optical communication network from a submarine deep in the ocean to a mother ship. Such networks are minimally invasive to the environment and highly flexible.

2. Design and Modeling

This chapter deals with the design of the SUUV, the different parts and the construction of it. Further, the kinematic model of the robot is introduced, whereby several assumptions are made which simplify the kinematic model of the swimmer. The design and modeling introduced in this chapter is based on [1] and the DGCMG firstly is published in the work of Mohsen Saadat in [3].

2.1 Swimmers Design

The design of the swimmer is developed by Mohsen Saadat, a PhD students at the Department of Mechanical Engineering, University of Minnesota, Minneapolis, USA. It is composed of a symmetric shaped body, two external thrusters (T_1 and T_2) and a double gimbal system with a high speed wheel located in the center of the swimmer. Both external thruster are responsible for the translational movements of the swimmer. The high-speed wheel of the double gimbal system creates an inertia platform which enables the rotation of the swimmer in 3D space. Figure 2-1 below shows the design of the swimmer.

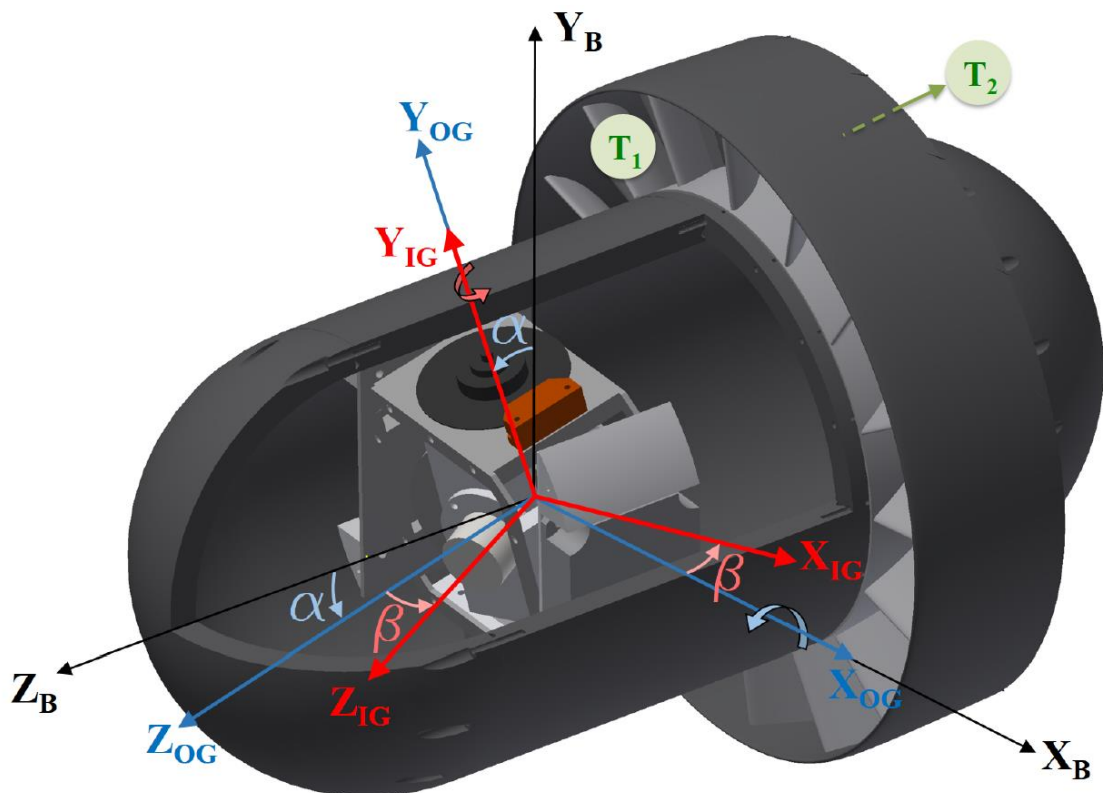


Figure 2-1: Super-Maneuverable Unmanned Underwater Vehicle Design

The external thrusters are counter rotating; this generates an additional degree of freedom for the control of the swimmer's orientation. The center of the robot is a Double

Gimbal Control Moment Gyro (DGCMG). It consists of a high-speed wheel in the center. This wheel rotate with up to 12,000 rotations per minute (RPM) and creates an inertia platform. Two gimbals enable to reorientate the gyro in 3D space and therefore rotate the fuselage of the swimmer. The DGCMG can be seen in Figure 2-2.

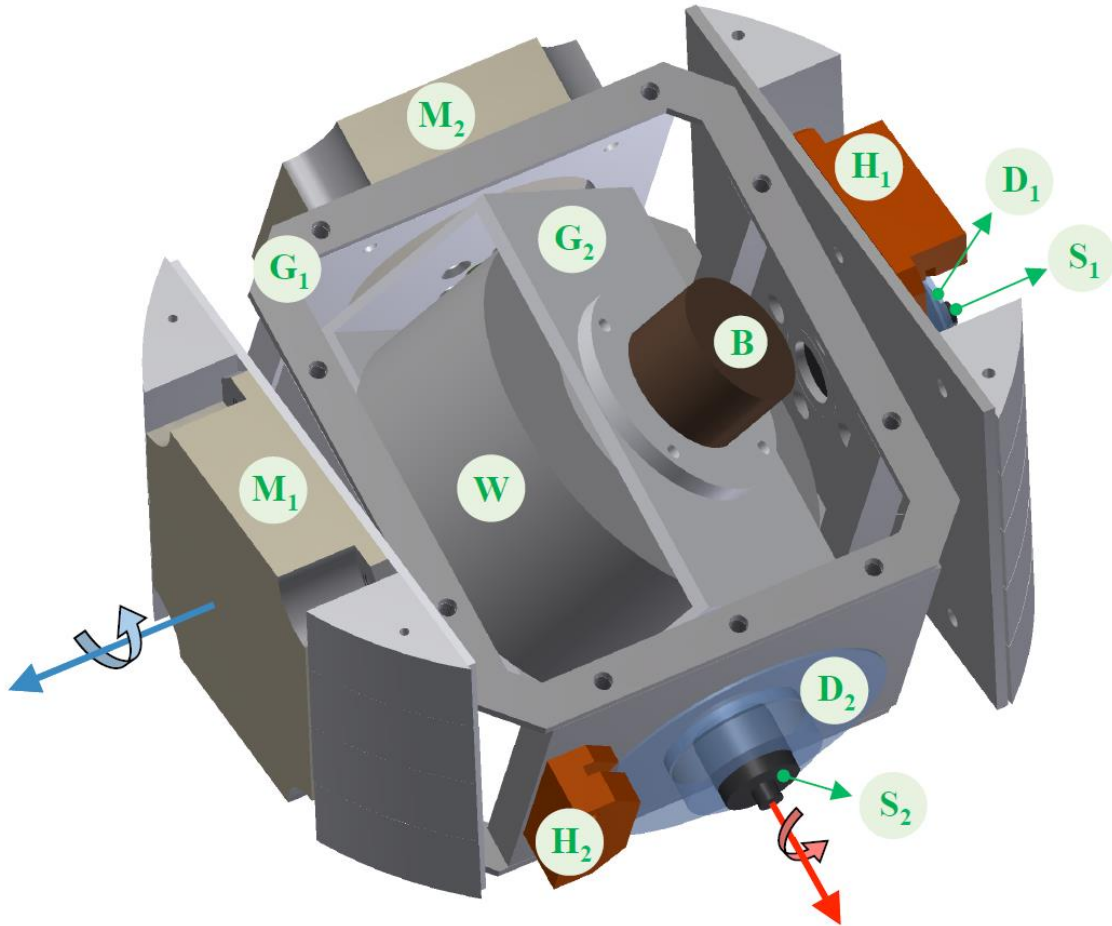


Figure 2-2: Double Gimbal Control Moment Gyro (DGCMG)

The center wheel W is proposed by a high-speed brushless DC motor B . The inner gimbal (IG) G_2 provides the housing for the center wheel, whereby the gyro is fixed to the inner gimbal. This gimbal is driven by the stepper motor M_2 and can be fully rotated. A slice ring S_2 passes through the motor signals from the center to the outer gimbal (OG) G_1 . An optical encoder H_2 with its high resolution encoder disk D_2 is used to measure the position of the inner gimbal. A second slide ring S_1 connects the inner gimbal through the outer gimbal G_1 with the swimmers fuselage, where the electronic is located. The second stepper motor M_1 enables the rotation of the outer gimbal and the second optical encoder H_1 with its encoder disk D_1 measures the position of the outer gimbal respectively. As mentioned previous, the center wheel rotates with up to 12,000 RPM. The rotation axis of the outer and inner gimbals are always perpendicular to each other. The inner gimbal has two rotational degrees of freedom and the outer gimbal one degree of freedom with respect to the swimmer's body. The center wheel has three rotation

degrees of freedom with respect to the SUUV's body. With respect to the inertial frame, the body itself has six degrees of freedom, three translational and three rotational. This leads to the assumption that the SUUV is underactuated in case of reorienting the swimmers body. Three independent degrees of freedom would be needed to control the orientation of the Unmanned Underwater Vehicle (UUV). The double gimbal system can just provide two degrees of freedom whereby both degrees of freedom are non-linear dependent of each other. Therefore, the external thrusters are needed to deliver an additional degree of freedom and to make the system fully controllable.

2.2 Dynamic Model

The gyro center wheel rotates with a constant velocity of up to 12,000RPM. When one of the two gimbals rotate and change the position of the center wheel, the change of the wheels angular momentum vector introduce a reaction force and a gyro force to the swimmers body. This force is transferred to the body where the double gimbal system is mounted to the fuselage, yielding a rotation of the body. Let us assume that the wheel spins with an infinite angular velocity, then the angular momentum vector will be invariant in space. In reality, the max. angular velocity of the wheel is limited and the wheel will rotate depending on the power applied by the two stepper motors M_1 and M_2 . In the case of an infinite angular speed the wheel can be seen as a inertia platform for the reorientation of the swimmer. However, in reality the change of the angular momentum vector will be smaller because of its high angular momentum. Finally, the SUUV shall be reoriented by tilting the spin axis of the center wheel without changing its angular velocity.

2.2.1 Reference Frames

The coordinate reference frames are introduced in Figure 2-1. Three coordinate systems are needed to represent the dynamic of the UUV. The dynamic of the swimmer is a combination of four connected ridged bodies, whereby the center wheel is always attached to the inner gimbal. Therefore, three coordinate frames are needed to represent the dynamics of the swimmer. These frames are shown in Figure 2-3 below specifically.

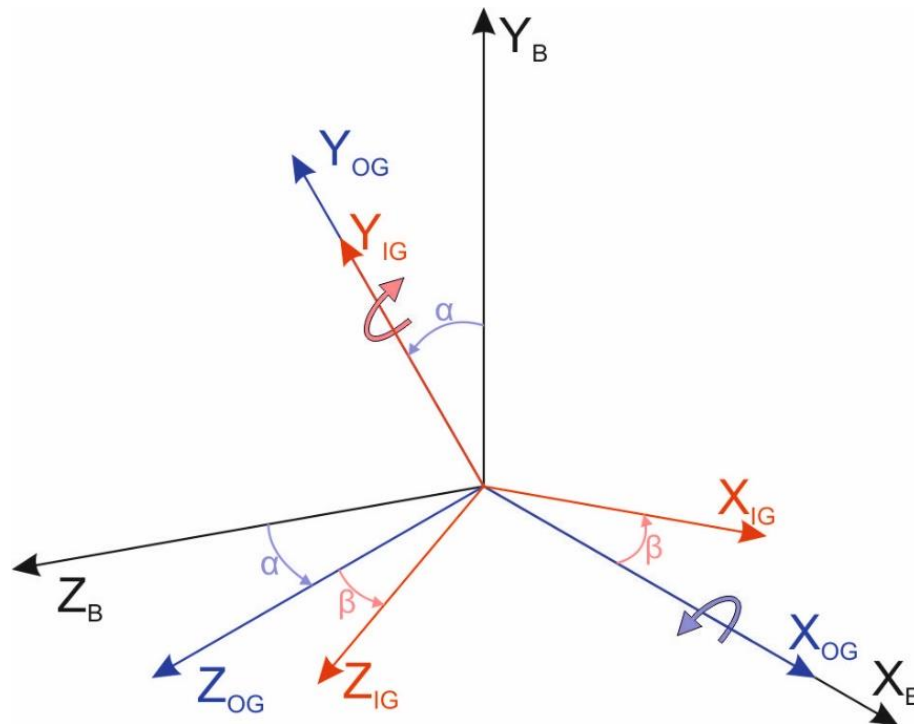


Figure 2-3: Double Gimbal Control Moment Gyro – Coordinate Systems

The body coordinate frame (X_B , Y_B and Z_B) is attached to the body of the swimmer, whereby the Z_B -axis is aligned with the force vector of the two thrusters $T1$ and $T2$. The outer gimbal is attached to the body coordinate frame with the outer gimbal coordinate system (X_{OG} , Y_{OG} and Z_{OG}). It is aligned with the X_B -axis of the body frame. The rotation of the outer gimbal with respect to the body frame is defined by the angle α . To the outer gimbal the inner gimbal coordinate frame (X_{IG} , Y_{IG} and Z_{IG}) is attached. The Y_{IG} -axis of the inner gimbal is aligned with the Y_{OG} -axis of the outer gimbal coordinate frame. The rotation from the inner gimbal with respect to the outer gimbal is described by the angle β . No further coordinate frame is needed for the center wheel because its rotation is already aligned with the inner coordinate frame. Using these three coordinate systems the dynamics of the swimmer can be described.

2.2.2 Fundamental Equations

This chapter deals with the derivation of the fundamental equations of the dynamic model. In a first step, the inertias of the four connected ridged bodies are defined. In a further step, the angular velocities and the angular momentums are defined. Thirdly, Newton's formulism is used to derive the body torques. Finally, using unit quaternions and several assumptions, the final dynamic model can be derived.

2.2.2.1 Body Angular Momentums

The body of the swimmer consists of four connected ridged bodies. The inertias of the individual bodies are defined by

$$\begin{aligned}
 [{}^B \mathbf{I}_B] &= [\text{diag}]_{3 \times 3} \\
 [{}^{OG} \mathbf{I}_{OG}] &= [\text{diag}]_{3 \times 3} \\
 [{}^{IG} \mathbf{I}_{IG}] &= [\text{diag}]_{3 \times 3} \\
 [{}^{IG} \mathbf{I}_W] &= [\text{diag}]_{3 \times 3}
 \end{aligned} \tag{2.1}$$

Whereby ${}^a \mathbf{I}_b$ is the moment of inertia of the body b in the body coordinate frame a . These moments of inertia for B , OG , IG and W are defined for the body, the outer gimbal, the inner gimbal and the center wheel respectively. The moment of inertia of the center wheel ${}^{IG} \mathbf{I}_W$ is defined in the inner gimbal coordinate frame. The angular momentum of each body is defined by

$$\begin{aligned}
 {}^B \mathbf{H}_B &= [{}^B \mathbf{I}_B] \cdot {}^B \boldsymbol{\omega}_B \\
 {}^{OG} \mathbf{H}_{OG} &= [{}^{OG} \mathbf{I}_{OG}] \cdot {}^{OG} \boldsymbol{\omega}_{OG} \\
 {}^{IG} \mathbf{H}_{IG} &= [{}^{IG} \mathbf{I}_{IG}] \cdot {}^{IG} \boldsymbol{\omega}_{IG} \\
 {}^{IG} \mathbf{H}_W &= [{}^{IG} \mathbf{I}_W] \cdot {}^{IG} \boldsymbol{\omega}_W
 \end{aligned} \tag{2.2}$$

Whereby ${}^a \mathbf{H}_b$ is the angular momentum of body b in the coordinate system a . \mathbf{I} is the moment of inertia defined in (2.1). ${}^a \boldsymbol{\omega}_b$ is the angular velocity of body b in the a coordinate frame. By definition of the coordinate systems in chapter 2.2.1, the angular velocities are defined by

$$\begin{aligned}
 {}^{OG} \boldsymbol{\omega}_{OG} &= [\dot{\alpha} \quad 0 \quad 0]^T \\
 {}^{IG} \boldsymbol{\omega}_{IG} &= [0 \quad \dot{\beta} \quad 0]^T \\
 {}^{IG} \boldsymbol{\omega}_W &= [0 \quad 0 \quad \Omega]^T
 \end{aligned} \tag{2.3}$$

$\dot{\alpha}$ is the angular velocity of the outer gimbal in the outer gimbal coordinate system. $\dot{\beta}$ is defined as the angular velocity of the inner gimbal in the inner gimbal coordinate frame and Ω is the angular velocity of the center wheel with respect to the inner gimbal.

The total angular momentum of the body is the sum of the individual angular momentums of the four connected rigid bodies, defined in the body coordinate frames.

$${}^B \mathbf{H}_G = {}^B \mathbf{H}_B + {}^B \mathbf{H}_{OG} + {}^B \mathbf{H}_{IG} + {}^B \mathbf{H}_W \tag{2.4}$$

The total angular momentum ${}^B \mathbf{H}_G$ is defined in the body coordinate frame around the center of mass G .

2.2.2.2 Body Torques

Using Newton's formulism, to calculate the torque from the depending angular momentums, the rigid-body dynamic of the swimmer is governed by the following equation. Whereby, external drag forces and torques are neglected and just the torque of the two external counter rotating thrusters are taken into account, yielding:

$$\mathbf{T}_{\text{ext}} = \frac{d\mathbf{H}_G}{dt} \tag{2.5}$$

Where \mathbf{T}_{ext} is the external torque about the swimmers central mass and defined by $\mathbf{T}_{ext} = [0 \ 0 \ M]^T$. Further, M is the torque generated by the thrusters. As defined above \mathbf{H}_G is the total angular momentum around the center of mass G . Deriving the equation (2.5) with respect to the time, yields to:

$$\mathbf{T}_{ext} = \dot{\mathbf{H}}_G + \omega^x \cdot \mathbf{H}_G, \quad (2.6)$$

whereby ω^x is a skew-symmetric angular velocity matrix and defined as following:

$$\omega^x = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_3 & \omega_2 & 0 \end{bmatrix}. \quad (2.7)$$

The indices ω_i ($i = 1, 2, 3$) of equation (2.7) correspond to the axis (x, y, z) of the angular velocity vector of the swimmer respectively.

2.2.3 Coordinate System Rotations

The total angular momentum around the center of mass ${}^B\mathbf{H}_G$ introduced in equation (2.4) is based on the body coordinate frame. Further, all other bodies have to be in the same coordinate system. Euler angles are used to rotate the coordinate frames from their origin to the destination coordinate frame. The Euler rotation for the inner gimbal IG into the outer gimbal OG coordinate frame is defined by the angle β , yielding:

$$[{}^{OG}\mathbf{R}] = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \quad (2.8)$$

and the rotation matrix from the outer gimbal OG to the body frame B is defined by the angle α , yielding:

$$[{}^{OG}\mathbf{R}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}. \quad (2.9)$$

Regarding the definitions made in chapter 2.2.1 the inner gimbal IG rotates around the y -axis of the outer gimbal OG with the angle β and the outer gimbal OG rotates around the x -axis of the body B with the angle α . The rotation matrixes are define with $[\mathbf{R}]$, which gives the rotation from coordinate frame a to coordinate frame b . This leads to the new form for the individual angular momentums defined in the body coordinate frame:

$$\begin{aligned} {}^B\mathbf{H}_{OG} &= [{}^{OG}\mathbf{R}] \cdot {}^{OG}\mathbf{H}_{OG} \\ {}^B\mathbf{H}_{IG} &= [{}^{OG}\mathbf{R}] \cdot [{}^{IG}\mathbf{R}] \cdot {}^{IG}\mathbf{H}_{IG} \\ {}^B\mathbf{H}_W &= [{}^{OG}\mathbf{R}] \cdot [{}^{IG}\mathbf{R}] \cdot {}^{IG}\mathbf{H}_W \end{aligned} \quad (2.10)$$

Inserting these equations into (2.4) yields to:

$${}^B\mathbf{H}_G = {}^B\mathbf{H}_B + [{}^{OG}\mathbf{R}] \cdot {}^{OG}\mathbf{H}_{OG} + [{}^{OG}\mathbf{R}] \cdot [{}^{IG}\mathbf{R}] \cdot {}^{IG}\mathbf{H}_{IG} + [{}^{OG}\mathbf{R}] \cdot [{}^{IG}\mathbf{R}] \cdot {}^{IG}\mathbf{H}_W \quad (2.11)$$

The total angular momentum therefore is defined as the sum of the individual angular momentums and its rotations to the body coordinate frame. To rotate the outer gimbal to the body frame, one rotation is needed. From the inner gimbal to the body frame, two rotations are needed.

2.2.4 Assumptions

The dynamic of the swimmer is too complex to implement it in a real-time agility controller. Therefore, several assumptions are made to simplify the dynamic model of the UUV. This chapter contains these assumptions.

- (I) The angular momentum of the gimbals are much smaller than the angular momentum created by the center wheel. Thus, equation (2.4) can be reduced to:

$${}^B\mathbf{H}_G \cong {}^B\mathbf{H}_F + {}^B\mathbf{H}_W, \quad (2.12)$$

whereby the angular momentums \mathbf{H} are represented in the body frame ${}^B(\dots)$. The total angular momentum ${}^B\mathbf{H}_G$ is a combination of the angular momentum of the wheel ${}^B\mathbf{H}_W$ and the angular momentum of the fuselage ${}^B\mathbf{H}_F$ with respect to the swimmers center of mass.

- (II) The high-speed center wheel rotates much faster as the gimbals and therefore has a much higher angular momentum. Thus, the angular momentum components of the gimbals in direction of the wheel are neglected, expressed by:

$${}^B\mathbf{H}_G \cong [{}^B_{IG}\mathbf{R}] \cdot {}^{IG}\mathbf{H}_G = [{}^B_{IG}\mathbf{R}] \cdot [{}^{IG}\mathbf{I}_W] \cdot {}^{IG}\boldsymbol{\omega}_W \quad (2.13)$$

The rotation matrix from the inner gimbal to the body frame B is defined by $[{}^B_{IG}\mathbf{R}]$. The wheels inertia tensor $[{}^{IG}\mathbf{I}_W]$ and the angular velocity vector ${}^{IG}\boldsymbol{\omega}_W$ are defined in the inner gimbal frame IG . The angular velocity vector ${}^{IG}\boldsymbol{\omega}_W$ is defined in equation (2.3) and given by ${}^{IG}\boldsymbol{\omega}_W = [0 \ 0 \ \Omega]^T$, where Ω is the spinning frequency of the center wheel.

- (III) Regarding the axisymmetric shape of the fuselage of the swimmer and because its angular velocity components are smaller than the angular velocity Ω of the wheel during reorientation maneuvers, it can be assumed that terms regarding the product of the body's angular components can be neglected,

$$\boldsymbol{\omega}^x \cdot {}^B\mathbf{H}_F = \boldsymbol{\omega}^x \cdot [{}^B\mathbf{I}_B] \cdot \boldsymbol{\omega} \approx 0 \quad (2.14)$$

with the moment of inertia tensor $[{}^B\mathbf{I}_B]$ of the swimmer defined in the body frame and $\boldsymbol{\omega}^x$ is the skew-symmetric angular velocity matrix of the body.

2.2.5 Governing Equations

The governing equations of the swimmer can be derived by applying the equations from chapter 2.2.2 and the three assumptions made before. The unit quaternions $(q_0, \mathbf{q}) \in \mathbb{R} \times \mathbb{R}^3$ are used to describe the orientation of the swimmer. For a more detailed explanation of the quaternions, definition and operations see [9]. This helps to avoid singularities and gimbal locking in the dynamics and is needed in a further step for the proper control of the model. This yields the final equations of the full dynamic model:

$$\dot{\mathbf{q}} = \frac{1}{2}(\mathbf{q}^x \cdot \boldsymbol{\omega} + q_0 \boldsymbol{\omega}) \quad (2.15)$$

$$\dot{q}_0 = -\frac{1}{2}\mathbf{q}^T \cdot \boldsymbol{\omega} \quad (2.16)$$

$$\dot{\boldsymbol{\omega}} = \mathbf{F}_{(\alpha,\beta)} \cdot \boldsymbol{\omega} + \mathbf{G}_{(\alpha,\beta)} \cdot \mathbf{U} \quad (2.17)$$

Three matrixes are contained in equation (2.17), whereby $\mathbf{U} = (\dot{\alpha} \quad \dot{\beta} \quad M)^T$ is the control vector containing the angular velocity $\dot{\alpha}$ of the outer gimbal, the angular velocity $\dot{\beta}$ of the inner gimbal and the resultant torque M generated by the thrusters. The angular velocity $\boldsymbol{\omega}$ is defined in the body-fixed reference frame. The skew-symmetric matrix \mathbf{q}^x is formed by the elements of $\mathbf{q} = (q_1, q_2, q_3)^T$. The matrixes \mathbf{F} and \mathbf{G} are defined by:

$$\mathbf{F}_{(\alpha,\beta)} = J_W \Omega [{}^B \mathbf{I}_B]^{-1} \begin{bmatrix} 0 & -c_\alpha c_\beta & -s_\alpha c_\beta \\ c_\alpha c_\beta & 0 & -s_\beta \\ s_\alpha c_\beta & s_\beta & 0 \end{bmatrix} \quad (2.18)$$

$$\mathbf{G}_{(\alpha,\beta)} = J_W \Omega [{}^B \mathbf{I}_B]^{-1} \begin{bmatrix} 0 & -c_\beta & 0 \\ c_\alpha c_\beta & -s_\alpha s_\beta & 0 \\ s_\alpha c_\beta & c_\alpha s_\beta & \frac{1}{J_W \Omega} \end{bmatrix} \quad (2.19)$$

The moment of inertia tensor of the wheel J_W is defined around its spin axis, with its angular velocity Ω . The functions s_θ and c_θ ($\theta = \alpha, \beta$) correspond to $\sin \theta$ and $\cos \theta$ respectively.

3. Non-Linear Controller Design

As mentioned in chapter 2, the dynamic model of the swimmer is far too complex to use a linear controller for the attitude control. Therefore, a non-linear controller design is used to deal with the dynamics of the robot. A Lyapunov function is feasible to deal with high dynamic system behaviors. Further, the dynamics of the systems is coupled, which means that the rotation of one gimbal influences the rotation of the body in two axis, depending on the position of the other gimbal. An appropriate robust Lyapunov function with a Backstepping approach can deal even with such behaviors. The following chapter explains the theory behind both approaches and finally the controller design for the system dynamics is introduced.

3.1 Control Theory

This chapter deals with the theory behind the adaptive controller [10] [11]. The principle of the Lyapunov function is used to find a controller which is feasible to deal with the non-linearity of the system. This controller can stabilize the system and generate a robust and smooth state feedback control. Further, the adaptive Backstepping controller method is introduced.

3.1.1 Lyapunov Control

The Lyapunov stability criteria is a way to determine the stability of a system. In control theory, it can be used to determine if a controller is stabilizing a given system. This will be shown in the next chapter for the given dynamics.

3.1.1.1 Lyapunov Theory

Let us review the Lyapunov stability criteria on the example of a time-invariant system x and the dynamics

$$\dot{x} = f(x). \quad (3.1)$$

The Lyapunov theorem allows to prove that the system is stable. Therefore, a Lyapunov function $V(x)$ is needed. This function has to be defined positively in the region Γ near $x = 0$. Imagine the Lyapunov function V as an energy function which can never be negative and is just zero in the zero state. Rewriting this statement in equation form:

$$\dot{V}(x) = \frac{\partial V(x)}{\partial t} = \frac{\partial V(x)}{\partial x} \frac{\partial x}{\partial t} = \frac{\partial V(x)}{\partial x} f(x) \quad (3.2)$$

The Lyapunov theory defines that the solution is:

- *stable*, if $\dot{V}(x)$ is negative semi-definite in the region Γ

- *asymptotically stable*, if $\dot{V}(x)$ is negative defined in the region Γ
- *globally asymptotically stable*, if $V(x)$ is positive defined radially unbounded for all x , and if $\dot{V}(x)$ is negative defined for all x

In other words, this means that if the function is always decreasing it has to reach zero eventually, therefore there is no possibility that the system diverges. The system is stable.

3.1.1.2 Control Lyapunov Function

The definition of Lyapunov theorem can be used to determine a controller for a given system. The given system is extended with a Lyapunov function candidate and has to fulfill the Lyapunov theory for stability. One of the difficulties is to find the right Lyapunov function. Different procedures exist to evolve these kind of functions, which are explained in [10] and [11]. However, one of the most common ones is to take the error control function and use a power of it. Another one is to take the total energy function of the system. Let us explain the basic definition of the control Lyapunov function on the defined function (3.1) with the control input u :

$$\dot{x} = f(x, u) \quad (3.3)$$

The goal is to ensure the stability of the system, whereby a control law $u = \alpha(x)$ is used, which yields to:

$$\dot{x} = f(x, \alpha(x)) \quad (3.4)$$

Further, a Lyapunov function $V(x)$ is chosen, which has to fulfill $\dot{V}(x) < 0$. A straightforward approach to find the control law $\alpha(x)$ is to define a smooth positive definite and radially unbounded function $V(x)$ and then select $\alpha(x)$, which fulfills the following criteria:

$$\dot{V}(x) = V_x(x)f(x, \alpha(x)) < 0 \quad (3.5)$$

A Lyapunov function which satisfies these criteria, is called Control Lyapunov Function (CLF). An important fact is that a failure of the Lyapunov stability theorem does not mean that the equilibrium is stable or not. It just means that this Lyapunov function candidate does not fulfill the requirements for the Lyapunov stability. Further investigations regarding stability of the equilibrium or an enhancement of the Lyapunov function candidate, to satisfy the stability requirements, have to be done.

3.1.2 Backstepping

Another approach to find the Lyapunov function is the Backstepping method. This chapter will explain the fundamental theory behind. Further examples and explanations can be found in [10] and [11]. The Backstepping approach enables the control of a class

of non-linear functions with a lower triangular structure. This structure is solved in a recursive manner. Given is a second-order system with the form:

$$\begin{aligned}\dot{x} &= f(x) + g(x)\xi \\ \dot{\xi} &= u\end{aligned}\tag{3.6}$$

With the two state variables x and ξ as well as the control input u . The variable ξ is referred as a virtual control. The goal is to design a state feedback control law to stabilize the origin ($x = 0, \xi = 0$). The system in (3.6) is a combination of two systems and can be seen as a cascade connection of a system and an additional integrator. Let us suppose that the system can be stabilized by a smooth state feedback control law $\xi = \phi(x)$, with $\phi(0) = 0$, which yields to:

$$\dot{x} = f(x) + g(x)\phi(x)\tag{3.7}$$

Applying a smooth and positive defined Lyapunov function $V(x)$ yields to the Lyapunov stability criteria:

$$\frac{\partial V}{\partial x} [f(x) + g(x)\phi(x)] < 0\tag{3.8}$$

In a next step, the difference between the virtual control ξ and the desired value $\phi(x)$ is defined with:

$$z = \xi - \phi(x)\tag{3.9}$$

Finally, this results in:

$$\begin{aligned}\dot{x} &= [f(x) + g(x)\phi(x)] + g(x)z \\ \dot{z} &= u - \dot{\phi}\end{aligned}\tag{3.10}$$

This approach can be extended to higher order systems and follows always the basic three steps:

1. A virtual state and virtual control are introduced and the current state equation is rewritten in terms of these
2. The CLF is chosen for the new system and treated as the final stage
3. An equation for the virtual control is defined which finally stabilizes the CLF.

This procedure of moving the virtual states through the chain of integrators is called Backstepping.

3.2 Adaptive Controller Design

The criteria for the CLF and the Backstepping method is used for the final controller design for the dynamics of the swimmer. The dynamics is non-linear, which the Lyapunov theorem can deal with. The control inputs of the systems are defined with the velocities of the stepper motors and the two thrusters. Finally, the orientation of the system shall be controlled. This system is very similar to the system described in 3.1.2 and suitable

3. Non-Linear Controller Design

for a Backstepping approach. The goal is to control the orientation error to a small neighborhood around zero. The system is under-actuated without the additional input of the thruster; therefore, a continuous state feedback control is not feasible.

The non-linear Lyapunov control approach is introduced by the work of the authors of [1] and the software implementation is part of this thesis. To avoid singularities in the control algorithm the controller is implemented in unit quaternions. The definition of the error quaternion [12] $(e_0, \mathbf{e})^T \in \mathbb{R} \times \mathbb{R}^3$, with the desired orientation $(q_0^*, \mathbf{q}^*)^T \in \mathbb{R} \times \mathbb{R}^3$ and with the actual orientation $(q_0, \mathbf{q})^T \in \mathbb{R} \times \mathbb{R}^3$ is defined with:

$$\mathbf{e} = q_0^* \mathbf{q} - q_0 \mathbf{q}^* + q^x \cdot \mathbf{q}^* \quad (3.11)$$

$$e_0 = q_0 q_0^* - \mathbf{q}^T \cdot \mathbf{q}^* \quad (3.12)$$

Whereby, q^x is a skew-symmetric matrix and defined as follows:

$$q^x = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_3 & q_2 & 0 \end{bmatrix}. \quad (3.13)$$

The error dynamics of the attitude controller is defined in unit quaternion with:

$$\dot{\mathbf{e}} = \frac{1}{2} (\mathbf{e}^x + e_0 I_{3 \times 3}) \cdot \boldsymbol{\omega} \quad (3.14)$$

$$\dot{e}_0 = -\frac{1}{2} \mathbf{e}^T \cdot \boldsymbol{\omega} \quad (3.15)$$

With $I_{3 \times 3}$ is a 3×3 identity matrix and \mathbf{e}^x is a skew-symmetric matrix and defined as follows:

$$\mathbf{e}^x = \begin{bmatrix} 0 & -e_3 & e_2 \\ e_3 & 0 & -e_1 \\ -e_3 & e_2 & 0 \end{bmatrix}. \quad (3.16)$$

The error quaternion is subject of the following constrain:

$$\mathbf{e}^T \cdot \mathbf{e} + e_0^2 = 1 \quad (3.17)$$

Finally, it can be seen from (3.17) that if $e(t) = 0$ then $|e_0(t)| = 1$. Thus, the control objective is to drive $e(t)$ to zero. Shown in [1] the Lyapunov function candidate is chosen as the square of the error quaternion:

$$V = \frac{1}{2} \mathbf{e}^T \cdot \mathbf{e} \quad (3.18)$$

Substituting (3.14) and taking the timer derivative of (3.18) yields to:

$$\dot{V} = \frac{1}{2} e_0 \mathbf{e}^T \cdot \boldsymbol{\omega} \quad (3.19)$$

Using a smooth state feedback control $\boldsymbol{\omega} = \boldsymbol{\phi}_{(e, e_0)}$ with $\boldsymbol{\phi} = -k_1 e_0 \mathbf{e}$ the terms in (3.11) can be stabilized. Defining $k_1 > 0$ and the Lyapunov stability criteria yields to:

3. Non-Linear Controller Design

$$\dot{V} = -\frac{1}{2}e_0^2 e^T \cdot e < 0 \quad (3.20)$$

This shows that the time derivate is smaller than zero in every time step which proves that the Lyapunov function candidate is valid. Defining the Lyapunov function with the control error:

$$V_a = V_{(e)} + \frac{1}{2}(\omega - \Phi_{(e,e_0)})^T \cdot (\omega - \Phi_{(e,e_0)}) \quad (3.21)$$

Using equation (2.17) with its matrix definitions in (2.18) and (2.19) as well as taking the timer derivate of the equation above, yields to:

$$\dot{V}_a = \frac{\partial V}{\partial e} L \Phi + (\omega - \Phi)^T \cdot (\mathbf{F}_{(\alpha,\beta)} \cdot \omega + \mathbf{G}_{(\alpha,\beta)} \mathbf{U} - \dot{\Phi} + L^T \cdot e) \quad (3.22)$$

The matrix L is defined with $L = \frac{1}{2}(e^x + e_0 I_{(3 \times 3)})$. It can be seen that if $e \neq 0$ the right hand side of the equation above is always negative. Finally, the control signal U is defined with:

$$\mathbf{U} = \mathbf{G}^{-1} \cdot [-k_2(\omega - \Phi) + \dot{\Phi} - L^T \cdot e - \mathbf{F} \cdot \omega] \quad (3.23)$$

Which, leads to the final form of the Lyapunov function:

$$\dot{V}_a = -k_1 e_0^2 e^T \cdot e - k_2(\omega - \Phi)^T \cdot (\omega - \Phi) \quad (3.24)$$

The control Lyapunov function shows that the origin ($e = 0$ and $\omega = 0$) is asymptotically stable.

4. Electro-Mechanical Construction

The electro-mechanical construction of the SUUV contains the control of the Inputs/Outputs (I/O) as the communication with a higher control tool. This chapter describes the chosen Center Processing Unit (CPU), the used embedded system and the control of the I/Os of the system.

4.1 Embedded System

The embedded system is the center computing unit of the SUUV. Manly responsible for the communication with the I/O-Devices and MATLAB Simulink. Several requirements have to be fulfilled. The system has to be capable of the handling with several I/O devices such as motors, sensors or encoder. An easy way should be found to communicate with other systems as like a personal computer.

Because of these requirements, an Arduino embedded system [13] is chosen. Arduino provides different extensions as motor driver shields, Ethernet shields or fitting sensors. Additionally, a user-friendly development environment is provided. Different boards with different properties can be chosen, which is explained in the next subchapters.

4.1.1 Arduino Genuino Uno Board

The Arduino Genuino Uno board [14] is one of the first developed Arduino boards and based on the ATmega328P. The Uno was the first developed Arduino board and is used as a reference model for further Arduino generations. The ATmega328P has a 16MHz clock and the board operates at a 5V logic level. In total, 16 digital in- and outputs are provided, whereby six of them can be used as Pulse-Width Modulation (PWM) outputs. The layout of the board can be seen below.

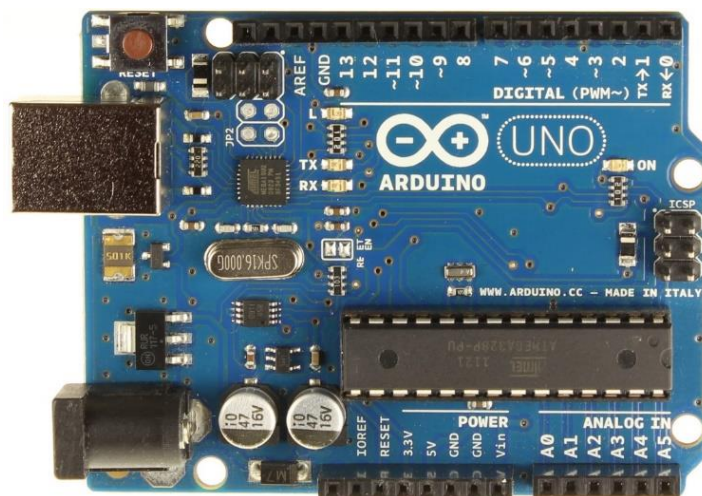


Figure 4-1: Arduino Genuino Uno

The Arduino Uno is one of the cheapest Arduinos and it is feasible because of its standard layout to operate with various Arduino extension boards.

4.1.2 Arduino Mega 2560 Board

The Arduino Mega 2560 Board [15] is powered by a processor with a 16MHz clock speed, has 54 digital I/O pins of which 15 provide PWM output. The operating voltage of the board is defined by 5V . Basically, the Mega can be seen as a bigger Arduino Uno. It is still capable to operate with most of the Arduino extension boards but provides more I/O possibilities. The layout of the Arduino Mega Board can be seen in Figure 4-2 below.

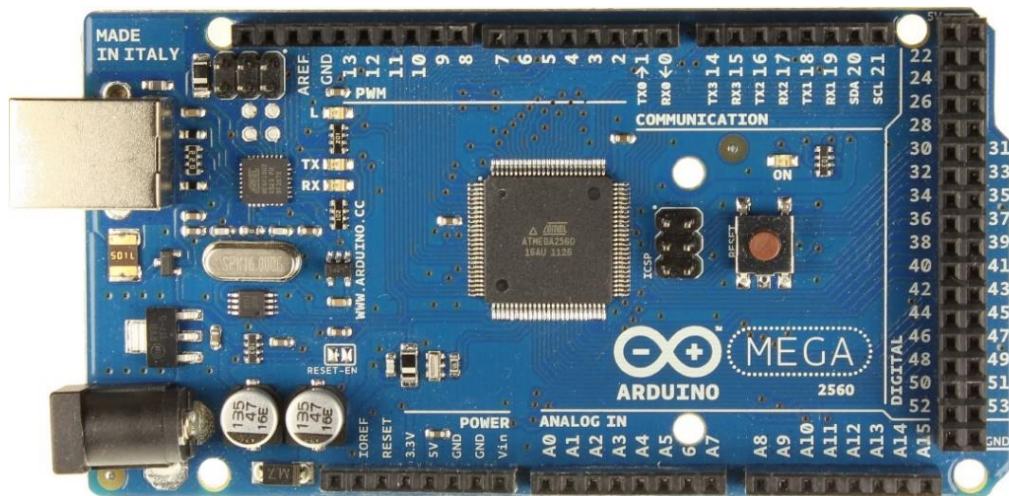


Figure 4-2: Arduino Mega 2560 Board

The Arduino Mega Board fulfills most of the requirements, but one of its limitations is the low clock speed. The project requires many different I/O devices, shields and other utilities. Because of this limitation, it has been decided to upgrade the Arduino Mega Board, which is described in the next chapter.

4.1.3 Arduino Due Board

The Arduino Due Board [16] operates at a voltage level of 3.3V , which includes all digital and analog I/O pins. An input signal of 5V at any pin could damage the board. Therefore, voltage level converter shall be used. The board is based on a 32-bit ARM core microcontroller with a clock speed of 84MHz . The schematic of the Arduino Due can be seen below.

4. Electro-Mechanical Construction

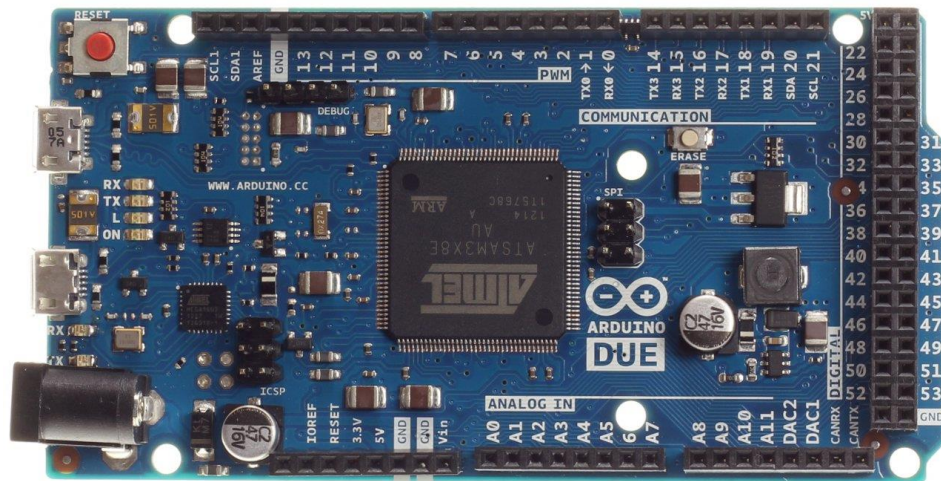


Figure 4-3: Arduino Due Board

The higher performance of the board allows the usage of multiple devices and other additional shields as like the Ethernet shield. Such components can require high performance and slow down the whole program if the clock speed is too low. The due board also allows applying interrupts at any pin and the usage of any pin as an analog output. This increases the flexibility of the board and allows a better applicability.

4.2 Arduino Open-Source Software (IDE)

Several developer tools can be used to program the Arduino boards. The Arduino open-source Software (IDE) [17] is the standard developer tool, which is provided by Arduino itself. The tool stands out by the simplicity of the layout and high usability. It is easy to learn and provides all necessary tools to develop your Arduino project. Figure 4-4 below shows the layout of the Arduino IDE.



```

Arduino_EthernetToMatlab_V4.0 | Arduino 1.6.9
Datei Bearbeiten Sketch Werkzeuge Hilfe

Arduino_EthernetToMatlab_V4.0 $
1 #include <Wire.h> // Arduino Wire Library for I2C Communication
2 #include <DueTimer.h> // Arduino DueTimer Library
3 #include <Servo.h> // Arduino Servo Library
4
5 #include <SPI.h> // SPI Library
6 #include <Ethernet2.h> // Ethernet Library
7 #include <EthernetUdp2.h> // Ethernet Udp Library
8
9 #include <Adafruit_Sensor.h> // Adafruit BNO0055 Orientation Sensor Libraries
10 #include <Adafruit_BNO055.h>
11 #include <utility/ImuMaths.h>
12
13
14 // *****
15 // ----- Pin Definition -----
16 // *****
17
18 // Define Stepper Motor Output Pins
19 const int stepper1_CLK_Pin = 34; // Stepper 1 Clock Pin
20 const int stepper1_Direction_Pin = 35; // Stepper 1 Direction Pin
21 const int stepper2_CLK_Pin = 32; // Stepper 2 Clock Pin
22 const int stepper2_Direction_Pin = 33; // Stepper 2 Direction Pin
23
24 // Define Brushless DC Motor Gyro Pins - Due Analog Pins 2 to 13
25 const int BLDC_Gyro_Enable_Pin = 7; // Brushless DC Motor Digital Output Pin - RPM
26 const int BLDC_Gyro_RPM_Pin = 6; // Brushless DC Motor Analog Output Pin - RPM
27
28 // Define Brushless DC Motor Rotor Thruster Pins
29 // Timer 5 -> D44, D45 & D46
30 const int BLDC_Thruster1_PWM_Pin = 50; // Brushless DC Motor 1 Pin - PWM
31 const int BLDC_Thruster2_PWM_Pin = 51; // Brushless DC Motor 2 Pin - PWM
32
33 // Matlab Arduino Ethernet Connection
34 // Enter a MAC address and IP address for your controller below.
35 // The IP address will be dependent on your local network:
36 byte mac[] = {0x90, 0xA2, 0xDA, 0x10, 0x6E, 0x8F};
37
38 IPAddress ip(192, 168, 1, 110);
39 const unsigned int localPort = 8888; // local port to listen on
40
41
42 // *****
43 // ----- Variable Definition -----
44 // *****
45
Code formatted for HTML has been copied to the clipboard.
Verify successful
Done in 11.565 seconds
Set boot flash true
CPU reset.
11 Arduino Due (Programming Port) auf COM8

```

Figure 4-4: Arduino Open-Source Software (IDE) Layout

4.2.1 Object-Oriented Programming

The used programming language is C resp. C++. The coding of the project is done in object-oriented programming. That means, classes and structures are generated to make the code more readable and compact. Further, this enables the possibility to use methods and parameters for multiple devices. For example if two similar motors need both the same functions and the same variables. The code below shows this case for two stepper motors, which need the same function and variables.

4. Electro-Mechanical Construction

```
1 // ----- Class - Stepper Motors -----
2 class stepper {
3
4     private:
5         const float MicroSteps = 32.0;
6         const float StpsPerRev = 200.0;
7
8     public:
9         float RPM_set;
10
11         unsigned long RpmToMicros()
12         {
13             // Conversion from Stepper RPM to Cycle Duration [ns] ->
14             // (((stepper1.RPM_set * 32.0 * 200.0) / 60.0) * 1000000.0)
15             return (1 / (abs(RPM_set) * MicroSteps * StpsPerRev / 60.0)) * 1000000;
16         }
17     };
18
19     stepper stepper1;
20     stepper stepper2;
```

Listing 4-1: Arduino Object-Oriented Programming

The class consists of private variables, which are just accessible class-internally. Public variables can be changed from the main loop or other functions. The class further contains a public function, which converts the stepper RPM into a microsecond time. Finally, two objects of the stepper class are generated, for each stepper motor a separate one. With the “this->”-command a class internal variable can be accessed; notice that this command is not always necessary. The class can be accessed in the setup and the main loop.

4.2.2 Interrupt Service Routines

Interrupt Service Routines (ISR) are needed if multiple tasks have to be handled simultaneously. Such ISR literally interrupt the main loop and process the interrupt handler. These functions are like other functions and are mainly used to do fast and time sensitive tasks such as the position read of a rotatory encoder. There are several ways to call ISR, for example internally by the Arduino intern timers or externally by a state change of an external pin.

Important thereby is that if the ISR interrupts the main loop, no other ISR can be called. Further, the serial communication is not reliable and functions such as the “delay()”-function is not working. This is because they are using the same internal timer. Thus, the interrupt handler functions should be as short and fast as possible. If the ISR and the main program manipulate global variables, they should be declared as “volatile” to ensure that they are updated correctly.

In the further chapters, interrupts are used to handle different tasks as like the position read of the optical encoder, the pulse generation for the stepper driver or for the PMW.

4.3 Stepper Motors

The two stepper motors build the core of the double gimbal system which is responsible for the reorientation of the robot. Therefore, special requirements on the stepper motors are given such as high precision, small design and high accuracy. The Sanyo Pancake Stepper Motor [18] fulfils all these requirements. The stepper motor can be seen below.

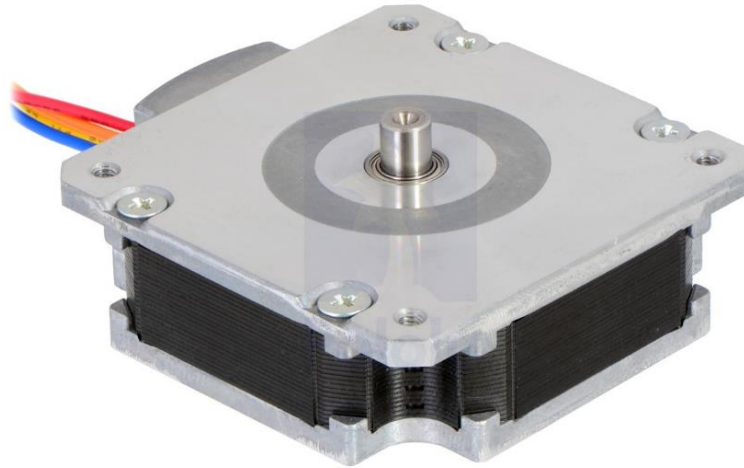


Figure 4-5: Sanyo Pancake Stepper Motor

The 1.8° step size and the possibility to drive the motor in microstepping mode enables a high accuracy for the positioning of the motor. The next chapters describe several methods which are used to control the stepper motor. Such methods are motor shields or different stepper motor driver.

4.3.1 Adafruit Motor Shield 2

The Adafruit Motor Shield 2 [19] for Arduino allows controlling the stepper motors directly with the Arduino extension shield. The easy handling of the shield and the provided libraries enable an easy setup of the shield and the stepper. The extension shield can be seen below.



Figure 4-6: Adafruit Motor Shield 2

The shield contains the complete electronic for the pulse generation and the stepper motors control. Up to two stepper motors can be controlled simultaneously. However, after several experiments and the usage of different libraries it has been shown that the shield does not fulfill the requirements in prospective to accuracy and rotational speed.

4.3.2 Stepper Motor Driver

Stepper motor driver are developer boards based on a microchips, which includes the necessary control electronic and the H-bridges for the current control for the stepper motors. There exist different boards based on different chips. The inputs for the board is for one a pulse signal, whereby the stepper motor perform for each pulse one-step. The second input is the direction for the stepper driver. The driver generates the corresponding input signals for the stepper motor. The next chapters describes two of the most common used drivers, including their advantages and disadvantages.

4.3.2.1 Polulu A4988 - Stepper Motor Driver

The Polulu Stepper Motor Driver A4988 [20] is one of the most common drivers available on the market. This stepper motor driver generates the output pulses for the two coils of the stepper motor. The layout is shown in Figure 4-7 below.

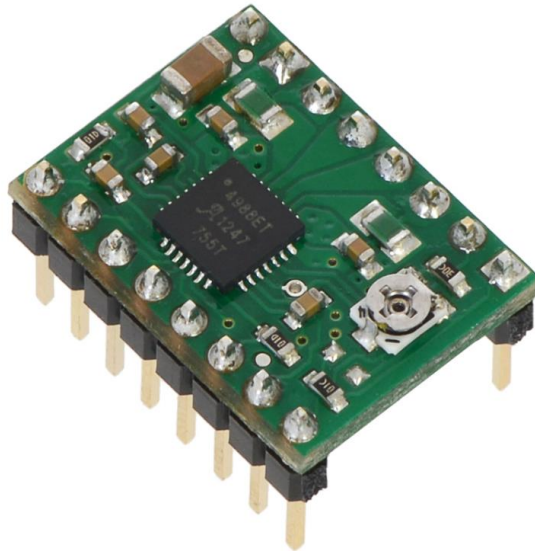


Figure 4-7: Polulu A4988 – Stepper Motor Driver

The board has to be fed by a pulse signal whereby one-pulse at the input results in one-step of the motor. The stepper motors can be driven in microstepping mode that means that each step is divided into 16 sub steps. This results to 3,200 steps per rotation at a 1.8° stepper motor. The microstepping mode increases the smoothness and the precession of the stepper motor. The driver can provide a Root Mean Square (RMS) current of 1A per coil. The standard wiring diagram of the stepper driver with the pancake stepper motor can be found in Figure 4-8.

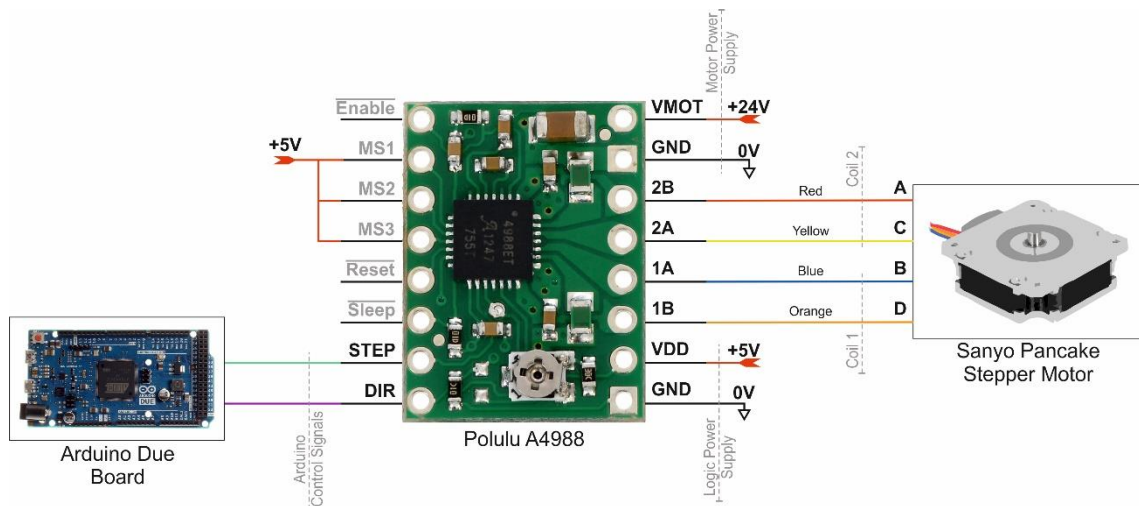


Figure 4-8: Sanyo Pancake Stepper Motor with Polulu A4988 Driver – Wiring Diagram

The control inputs MS1, MS2 and MS3 define the step mode of the motor. Whereby a high level at all three control inputs (MS1, MS2 and MS3) results in microstepping mode. A detailed description can be found in the datasheet [20]. Different approaches are used to generate the input step signal for the stepper driver, as described in the next chapter. The driver is fed with a 24V motor power supply signal and a 5V logic power supply. The

4. Electro-Mechanical Construction

two coils of the driver are connected with the four outputs of the driver. The Arduino control signals consists of the direction and the step signal.

Several tests showed that the Polulu A4988 driver cannot deliver the required torque to hold the gimbals at full gyro speed. Additional, the precision of the driver is not sufficient and the movements are not that smooth because of the relatively low maximal microstepping step size. Therefore, other drivers where tested.

4.3.2.2 Trinamic TMC2100 - SilentStepStick Stepper Motor Driver

The Trinamic TMC2100 SilentStepStick stepper motor driver [21] stick out with the high microstepping resolution. The driver itself is fed with a micro stepping signal with 16 micro-steps, the same as the Polulu A4988 driver. Figure 4-9 shows the new Trinamic driver.

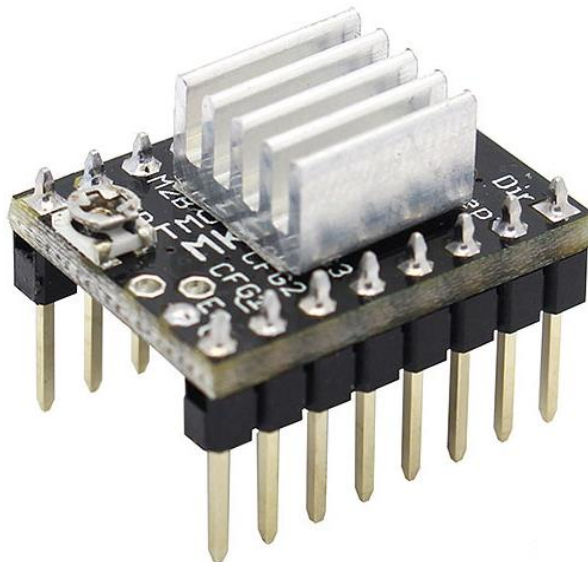


Figure 4-9: Trinamic TMC2100 – Stepper Motor Driver

The difference is that the Trinamic driver interpolate between the steps and therefore generate a virtual microstepping resolution of 256 micro-steps. This increases the smoothness and the torque at the stepper motor. Additionally, the driver can provide a higher RMS current of 1,2A per coil, in comparison to the Polulu A4988 driver, which delivers 1,0A per coil. Both driver have the same board layout . The wiring diagram for the driver can be found below.

4. Electro-Mechanical Construction

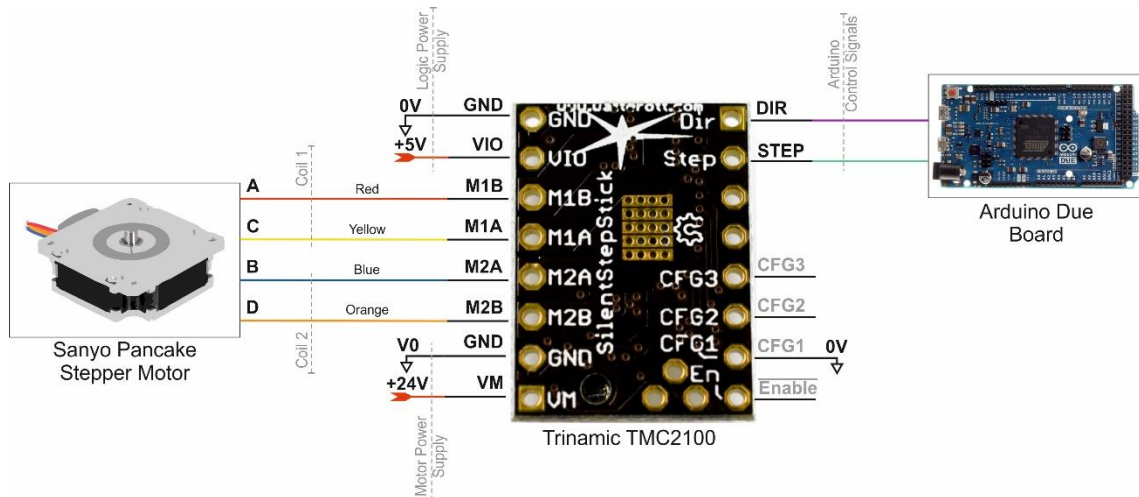


Figure 4-10: Sanyo Pancake Stepper Motor with Trinamic TMC2100 Driver – Wiring Diagram

Important is that the driver is mounted upside-down. Therefore, the pins are flipped and this enables to mount heat sinks, to cool the microchip. To enable the microstepping mode, with the virtual microstepping resolution of 256-sub steps, the CFG1 pin is connected to the ground and the CFG2 pins is not connected to any signal. The two coils of the pancake stepper motor are connected with the four outputs of the driver. As like as the Polulu A4988 driver, the Trinamic TMC2100 driver is fed with a motor power supply and a logic power supply signal. Experiment shows that the driver generates a high amount of heat. Therefore, it is recommended to mount reasonable heat sinks. Otherwise, the driver stops working at a max. temperature of 150°C . Several experiments show that the driver delivers a higher torque and better motion smoothness because of the virtual microstepping. Therefore, this driver is used for the final assembly. Further, the electrical circuits for both stepper driver introduced in the previous chapters can be found in appendix A in a better resolution.

4.3.2.3 Active Current Control

In the datasheet of the stepper motors can be seen that the operation voltage is about 5.9V at a maximum current of 1A per coil. The active current control of the stepper drivers enables to drive the stepper with 24V but the current is still limited to 1A per coil. This results too much smoother steps and a higher possible rotational speed. The active current control can be set by a potentiometer mounted on the stepper driver. For example, the active current control for the Trinamic TMC2100 [21] used in the project, can be set by the resistor seen in the picture below.



Figure 4-11: Trinamic TMC2100 – Stepper Motor Driver – Active Current Control

Because the Trinamic driver is mounted upside-down, the resistor can be adjusted through a hole in the board. The reference voltage for the regarding coil current can be found on the manufacturer homepage.

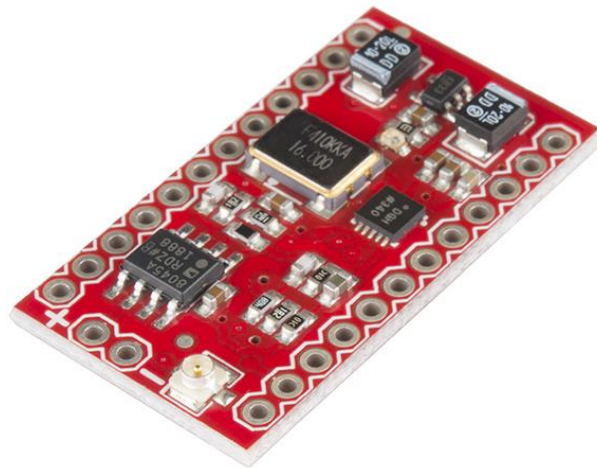
The implemented solutions allows driving the stepper motor with a maximal rotational velocity up to $200RPM$ and an extremely smooth acceleration.

4.3.3 Driver Input Signal Generation

As mentioned in the previous chapters, the stepper driver needs two input signals which are the signal for the direction of the motor and a pulse signal. For each pulse the stepper motor performs one step. The generation of the pulse signal is challenging, because in micro stepping mode with 16 micro-steps, a stepper motor with $200 \text{ steps per revolution}$ and a max. needed rotational speed of $60 RPM$ for the stepper motor; at least $3,200 \text{ pulses/s}$ are needed. Several methods exist to generate such a high amount of pulses which are described in the chapters below.

4.3.3.1 SparkFun MiniGen - Signal Generator Shield

The SparFun MiniGen Shield [22] (Figure 4-12) can generate sine, square, or triangle waves at up to $3MHz$ and communicate with Arduino via the Serial Peripheral Interface (SPI). This allows an extremely fast changing of the output state. The Arduino library provided by SparkFun allows an easy usage of the shield.



S

Figure 4-12: SparkFun MiniGen – Signal Generator Shield

However, after several attempts and consultation of the SparkFun support it was not possible to run the provided Arduino library at the Arduino Due board. Regarding of the consultation of the support team of SparkFun it was found that the Arduino library is not supported by the 32bit-processor architecture of the Arduino Due board.

4.3.3.2 Direct Output Register Toggling

The final approach is to toggle a digital output of the Arduino Due board and generate a square wave form. The Arduino Due has a clock speed of 84 MHz and should be able to toggle the output fast enough. However, the standard “digitalWrite”-command is not fast enough to switch the output at the needed frequency. Embedded systems allow the user to switch an output state by changing directly the output register which is extremely fast. The function to switch directly the output register can be found below:

```

1 // ----- Write Direct to the Arduino Due OUTPUT Register -----
2 // Write Direct to the Output Register of the Arduino -> Much Faster!!
3 // Needed for extremely fast Toggle Operations at the Stepper Pulse Generator
4 inline void digitalWriteDirect(int pin, bool val) {
5     if (val) g_APinDescription[pin].pPort -> PIO_SODR =
6         g_APinDescription[pin].ulPin;
7     else    g_APinDescription[pin].pPort -> PIO_CODR =
8         g_APinDescription[pin].ulPin;
9 }

```

Listing 4-2: Arduino Function – Direct Digital Write

This function can be used in a further step to change the output state in a timer interrupt. The timer interrupt is realized by the DueTimer library [23] which can handle all the internal timer of the Arduino Due. Further, two Arduino classes are generated which contain all the necessary variables and functions for the pulse generation. The ISR is linked in the setup with the “stepperClock”-function and updated with a frequency of 5,000Hz, as seen in the code below.

```

1 // ----- Stepper Motors Pulse Generator -----
2 // Attach Stepper Pulse Generators to Timer 6 & 7
3 Timer6.attachInterrupt(stepperClock1).setFrequency(5000);
4 Timer7.attachInterrupt(stepperClock2).setFrequency(5000);

```

4. Electro-Mechanical Construction

```
5 delay(250);
6 clockTimer1.Start = micros();
7 Timer6.start();
8 clockTimer2.Start = micros();
9 Timer7.start();
```

Listing 4-3: Stepper Driver – Attach stepperClock functions

In the timer interrupt a counter is implemented which measures the actual time and compares it with the calculated cycle duration (called interval time) of the desired rotational speed of the motor. The start time is the previous time from the last time step which is needed to calculate the time difference. A time trigger is set, which indicates that the cycle time is bigger than the interval time. Further, a toggle variable is used to save the previous output state and the output pin is toggled, which generates a rising or a falling signal, depending on the previous output state. The whole function can be found below.

```
1 // ----- Stepper Motors Pulse Generators -----
2 void stepperClock1() {
3
4     // Read Actual Time in Micros
5     clockTimer1.Actual = micros();
6
7     // Calculate if Next Timer Trigger is Set
8     clockTimer1.TimeTrigger = (clockTimer1.Actual - clockTimer1.Start) >=
9         clockTimer1.Interval;
10
11     // Block Motor if RPM = 0
12     if (clockTimer1.Interval != 0) {
13         // Check if Motor is ON and deltaTime is Larger than Time Interval
14         if (clockTimer1.ToggleStepper && clockTimer1.TimeTrigger) {
15             // Write Direct to Output Register of Arduino Due
16             digitalWriteDirect(stepper1_CLK_Pin, LOW);
17
18             // Save New Motor State
19             clockTimer1.ToggleStepper = 0;
20
21             // Turn Off Time Trigger
22             clockTimer1.TimeTrigger = 0;
23
24             // Save Actual Time -> Subtract Time Correction of Processing Time
25             clockTimer1.Start = micros();
26         }
27
28         // Check if Motor is OFF and deltaTime is Larger than Time Interval
29         if (~clockTimer1.ToggleStepper && clockTimer1.TimeTrigger) {
30             digitalWriteDirect(stepper1_CLK_Pin, HIGH);
31
32             clockTimer1.ToggleStepper = 1;
33
34             clockTimer1.Start = micros();
35         }
36     }
37 }
```

Listing 4-4: Stepper Driver – Direct Pulse Generation

It is important that all variables are defined as “volatile” because the compiler need to know that the variable state might be changed outside the main loop. Tests shows that this method can generate enough pulses to rotate both stepper motors with up to 200RPM.

4.4 Optical Encoders

Two US Digital EM1 Transmissive Optical Encoder Modules - EM1-2-1800-I [24] are used to measure the position and in a further step to calculate the angle of the inner and the outer gimbal. The encoders operate at a voltage level of 5V. The US Digital HUBDISK-2 2" Transmissive Rotary Disk [25] is used, which has a resolution of 7,200 *steps per revolution* which yields in a total resolution of 0.05 *degrees*. The figure below shows the optical encoder with its rotatory disk.



Figure 4-13: US Digital EM1 Transmissive Optical Encoder

Different methods are used to measure the ticks of the encoder. The encoder provides two channels (A and B) which are shifted to each other. By comparing both channels with each other, an upward or downward count can be calculated. The encoder is able to read the index position of the disk. Thus, the absolute position of the gimbals can be found. The following chapters show the different methods used to measure the gimbal positions.

4.4.1 Interrupt Encoder Read

As mentioned in chapter 4.2.2, Arduino provides the possibility to interrupt the main task when state changes on digital input pins appear. The fast rotation of the encoder disk generates high frequency pulses which the Arduino board has to measure. Thus, three interrupts, which detect a changing edge on the input pins, are used. Channel A and B give the change of the position of the encoder disk. The index channel is used to find the reference position of the referring gimbal. The setup of the interrupts for all three channels are implemented as follows:

4. Electro-Mechanical Construction

```
1 // ----- Generate Interrupts for Encoder -----
2 // Encoder Pin on Interrupt 0 - Pin encoder1_PinA (22)
3 attachInterrupt(digitalPinToInterrupt(encoder1_PinA),
4                 readEncoder1_PinA, CHANGE);
5
6 // Encoder Pin on Interrupt 1 - Pin encoder1_PinB (23)
7 attachInterrupt(digitalPinToInterrupt(encoder1_PinB),
8                 readEncoder1_PinB, CHANGE);
9
10 // Encoder Pin on Interrupt 2 - Pin encoder1_Index (24)
11 attachInterrupt(digitalPinToInterrupt(encoder1_PinIndex),
12                 readEncoder1_Index, CHANGE);
```

Listing 4-5: Arduino Encoder Direct – Attach Interrupt

The source code for the reading of the position is adapted from Arduino Playgrounds [26] and speed optimized. The high resolution of the encoder needs a fast processing of the ISR to make sure that each pulse is counted. The following sketch illustrates the interaction of channel A and B.

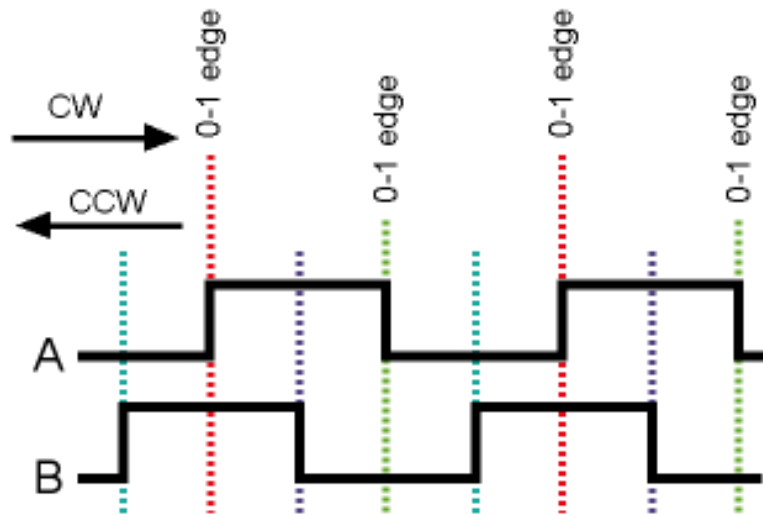


Figure 4-14: Rotary Encoder Channel Pulse Diagram

It can be seen that by checking a change state at channel A and the additional check of the high or low level of channel B the rotation direction can be predicted. The ISR detects a change on channel A or B and then directly reads the state of the regarding channel. To know if a rising or falling edge is detected. This state is saved and then compared with the saved state of the second channel. By saving the encoder state of both channels, just a single “read” command is needed per interrupt. This reduces the computational effort dramatically. Finally, both channels are compared and the upward or downward count is determined. This perception leads to the following source coder for channel A, B and the index of encoder 1:

```
1 // ----- Read Values from Encoder 1 -----
2 // Interrupt on A changing state
3 void readEncoder1_PinA()
4 {
5     // Check Actual Pin State
6     encoder1.A_set = digitalReadDirect(encoder1_PinA) == HIGH;
7     // and Adjust Counter + if A Leads B
```

4. Electro-Mechanical Construction

```
8   encoder1.Count += (encoder1.A_set == encoder1.B_set) ? +1 : -1;
9
10  }
11
12  // Interrupt on B changing state
13  void readEncoder1_PinB() {
14
15      // Check Actual Pin State
16      encoder1.B_set = digitalReadDirect(encoder1_PinB) == HIGH;
17      // and Adjust Counter - if B Leads A
18      encoder1.Count -= (encoder1.A_set == encoder1.B_set) ? +1 : -1;
19
20  }
```

Listing 4-6: Arduino Encoder Direct – Position Read

The interrupt on the index pin resets the counter value, as shown below.

```
1  void readEncoder1_Index()
2  {
3      // Reset Counter by Index Pin
4      encoder1.Count = 0;
5  }
```

Listing 4-7: Arduino Encoder Direct – Position Reset

It is important that the pin states are read directly from the output register because the normal read command is not fast enough to enable such high read cycles. Therefore, the “digitalReadDirect”-functions is implemented and can be seen below.

```
1  // ----- Write Direct to the Arduino Due INPUT Register -----
2  inline int digitalReadDirect(int pin)
3  {
4      return !(g_APinDescription[pin].pPort ->
5              PIO_PDSR & g_APinDescription[pin].ulPin);
6  }
```

Listing 4-8: Arduino Function – Direct Digital Read

The source code is tested and fast enough to count the *7,200 steps per revolution* of the encoder disk. First tests of the code showed that due to high vibrations of the gimbals at spinning gyro the encoder values leads to drift. This drift can be avoided by checking rising and falling changes of both channels. Further, a high performance code is needed as provided above. It is important that just on state is read per ISR, otherwise the Arduino is to slow and ISR calls get missed.

However, the final test of the system shows that the source code is fast enough to count the encoder value accurate. However, with all the other tasks such as the generation of the pulse signal for two stepper drivers, which is also done with ISR, the Arduino Due is on its computational limit. Thus, the Arduino is starting to miss ISR. Thus, it is decided to give this task to a special made encoder shield, as shown in the next chapter.

4.4.2 Robogaia Encoder Counter Shield

The Robogaia Encoder Counter Shield [27] is based on the LS7366R chip which is a counter chip and keeps track of the pulses delivered by channel A and B. The board can operate with up to three encoder simultaneous and provides various counter modes. It can be operated at a 5V or a 3.3V logic level. Because a lower logic level would reduce

4. Electro-Mechanical Construction

the sample rate of the chip, which is at 5V logic level up to 40kHz, it is decided to work with a 5V logic level. The encoder shield is seen in Figure 4-15.

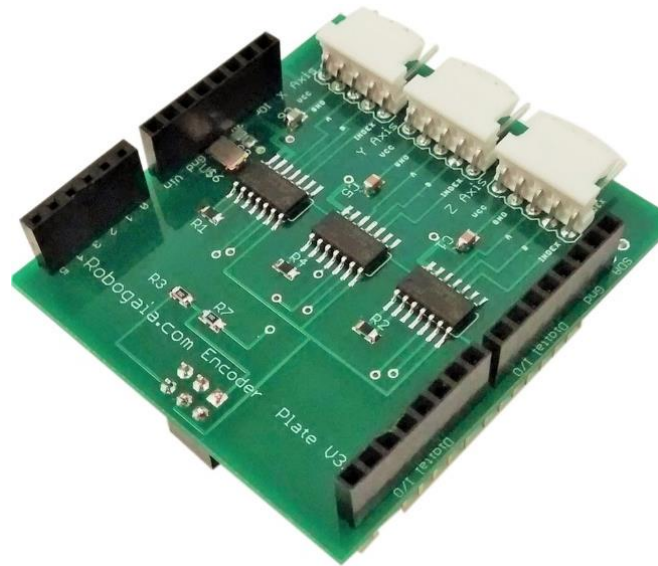


Figure 4-15: Robogaia Encoder Counter Shield

The communication of the encoder shield with the Arduino is realized via the SPI socket. The source code is based on the example code provided by the supplier of the shield. Basically, the operation mode of the LS7366R chip can be set by writing into the different chip register, which can be found in the datasheet. An Arduino class is generated, which combines all the functions of the encoder shield. The function for the initialization of the chip can be seen below:

```
1 // ----- Initialize RoboGaia Encoder Board -----
2 void LS7366_Init()
3 {
4     // Setup First Encoder Chip
5     digitalWrite(this->chipSelectPin, LOW);
6     SPI.transfer(0x88); // Write to MDR0 Register
7     SPI.transfer(0x03); // x4 Quadrature Mode
8     digitalWrite(this->chipSelectPin, HIGH);
9 }
```

Listing 4-9: Arduino Encoder Shield – Initialization

Setting the chip select pin to low enables the SPI communication at the regarding chip select pin. The operation modes is set and finally the SPI communication is disabled by writing the chip select pin to high. The “chipSelectPin”-variable is a public class variable and is set after generation of the encoder class. The source code to read the actual encoder value can be found below:

```
1 // ----- Read Encoder Values from Board -----
2 void readEncoderValues()
3 {
4     // Read Encoder Value
5     digitalWrite(this->chipSelectPin, LOW); // Enable SPI Communication on Pin
6     SPI.transfer(0x60); // Request Encoder Count
7     this->count1Value = SPI.transfer(0x00); // Read Highest Order Byte
8     this->count2Value = SPI.transfer(0x00);
9     this->count3Value = SPI.transfer(0x00);
10    this->count4Value = SPI.transfer(0x00); // Read Lowest Order Byte
```


4. Electro-Mechanical Construction

```
11  digitalWrite(this->chipSelectPin, HIGH); // Disable SPI Communication on Pin
12
13  this->Count = ((long)this->count1Value << 24) + ((long)this->count2Value <<
14  16) + ((long)this->count3Value << 8) + (long)this->count4Value;
15
16
17  // Calculate Angle from Actual Counter Value
18  this->Angle = (float)this->Count / this->encoderStpsPerRot * 360;
19
20  // Invert Negative Encoder Angles
21  if (this->Angle < 0) {
22  this->Angle = 360 + this->Angle;
23  }
24 }
```

Listing 4-10: Arduino Encoder Shield – Read Encoder Value

Writing a value request command to the regarding counter chip forces the chip to send the actual encoder value in four-byte format via SPI. These bytes can be read and combined by byte shift to the final encoder value. Afterwards the angle can be calculated and some value limitations are adjusted.

The reset of the encoder value is done by a “reset”-variable, which is a class external flag triggered by an ISR. In every main loop iteration the flag is checked and if the case the encoder counter is reset. The code can be found below.

```
1  // ----- Reset Encoder Values -----
2  void clearEncoderCount()
3  {
4      if (this->Reset) {
5          // Set Encoder1s Data Register to Zero
6          digitalWrite(this->chipSelectPin, LOW); // Begin SPI Conversation
7          SPI.transfer(0x20); // Reset Counter
8          digitalWrite(this->chipSelectPin, HIGH); // Terminate SPI Conversation
9
10         // Reset Clear Trigger
11         this->Reset = false;
12     }
13 }
```

Listing 4-11: Arduino Encoder Shield – Reset Counter

The final test shows that there is a drift of the encoder value even if vibrations at the gimbals occur because of the high speed gyro. Further, the computational effort of the Arduino is decreased.

4.5 Brushless DC Motor – Gyro

The center wheel of the double gimbal gyro system is actuated by the Anaheim Automation Brushless DC (BLDC) Motor [28]. The motor is capable to rotate with a velocity up to 12,000 *RPM*. The speed control of the motor is done by the Anaheim Automation Brushless Speed Controller Board [29]. The speed controller and the brushless DC motor can be seen in the figure below.

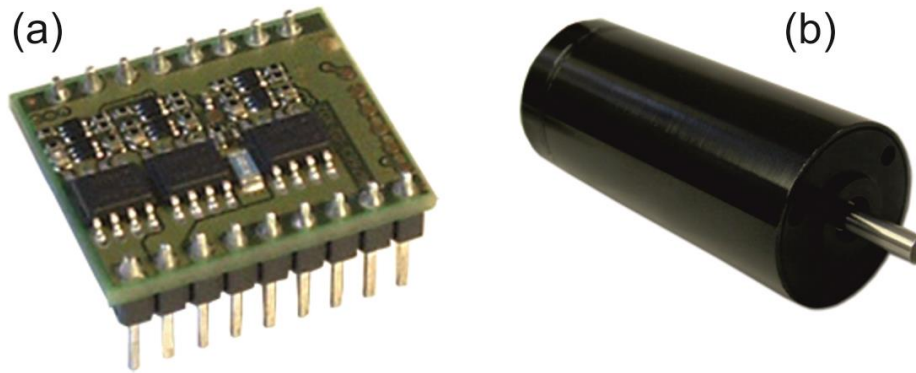


Figure 4-16: Speed Controller (a) and Brushless DC Motor (b)

The speed controller operates at a max. control voltage of 5V and a power supply of 24V for the brushless DC motor. The figure below shows the wiring of the BLDC motor with the speed controller. The output pins of the controller W1, W2 and W3 are connected with the Phase A, B and C of the motor respectively. The hall A, B and C wires of the motor are connected with the hall sensor 1, 2 and 3 of the controller respectively. Hall power and hall ground is connected with the hall power supply of the driver. However, the standard color of the motor wires can be seen in the figure below. Further, the electric circuit can be found in a higher resolution in appendix A.

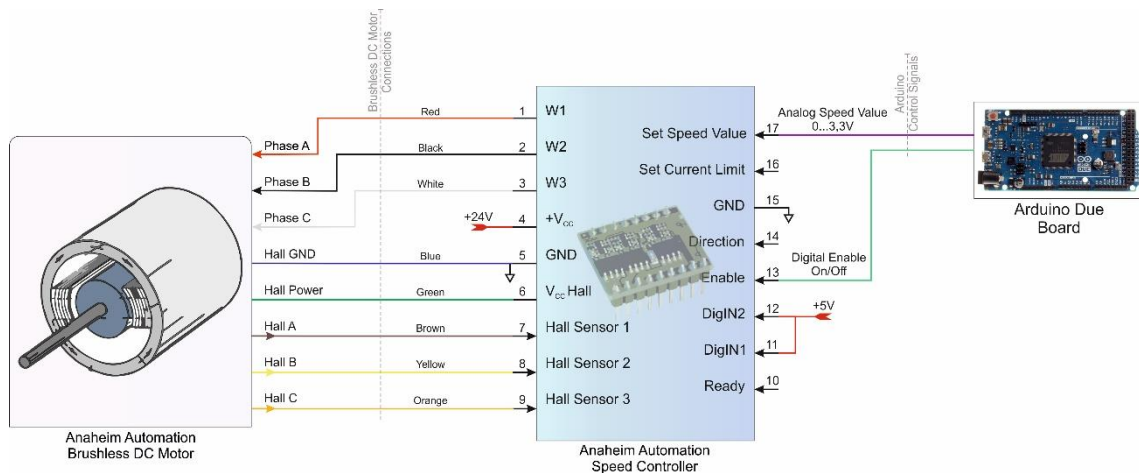


Figure 4-17: Brushless DC Motor with Speed Controller – Gyro – Wiring Diagram

By changing the analog voltage level at pin #17 (Set Speed Value) the velocity of the motor can be controlled. A voltage between 2.4V – 28V at the enable pin starts the brushless motor. Digital Input 1 and 2 at pin #11 and pin #12 respectively adjust the maximal motor velocity.

To enable a smooth acceleration of the motor, which is necessary because of the high inertia of the gyro and the high rotational velocity, a start-up procedure is programmed. The equation below, allows the conversion from the expected rotational speed to the needed input voltage at the controller board:

$$U_{out} = \left(\frac{(n - n_{min})}{(n_{max} - n_{min})} \cdot 4.9V \right) + 0.1V \quad (4.1)$$

The n_{min} and n_{max} values are depending on the control configuration, set by the two digital inputs. As mentioned, the motor velocity is controlled by a voltage between 0V to 5V at pin #17 of the driver. This signal is generated by an analog output of the Arduino Due board, which has a max. output voltage of 3.3V. Therefore, pin #11 and pin #12 are connected to 5V to enable a max. velocity of 20,000RPM at a analog voltage of 5V and motor velocity around 12,000RPM at a max. Arduino Due output voltage of 3.3V.

4.6 Brushless DC Motor – Thrusters

Two thrusters are used for the translational maneuvers of the swimmer. Further, these thrusters act as an additional control input for the system. Two BlueRobotics T100 Thruster [30] provide enough thrust to propose the swimmer, Figure 4-18.

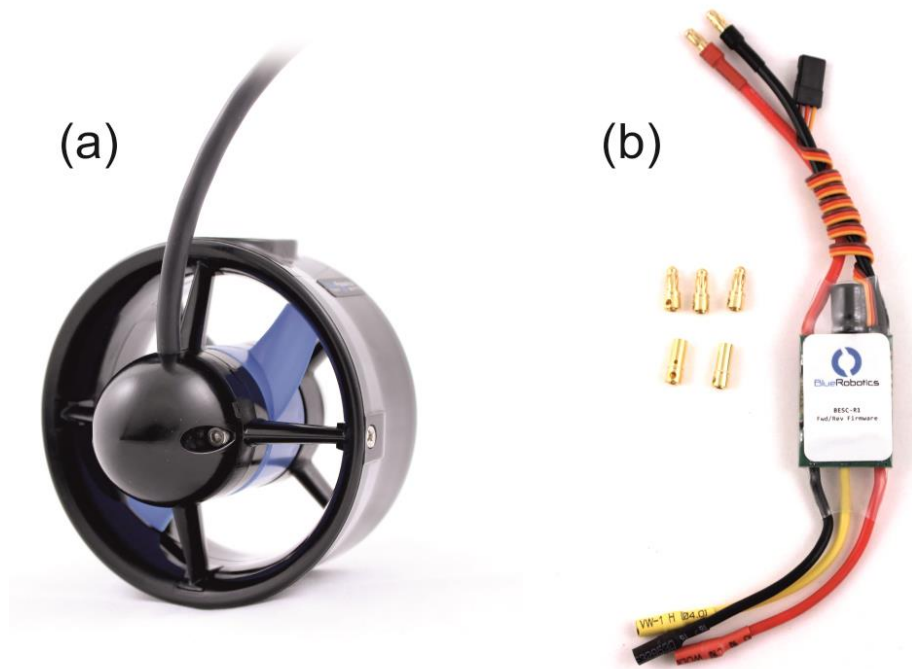


Figure 4-18: BlueRobotics T100 Thruster (a) and ESC Motor Controller (b)

The thrusters are actuated by waterproof brushless DC motors. The control of the two motors is done by the BlueRobotics ESC Controller [31]. The controllers are capable to provide up to 30A. The output velocity of the thrusters depend on the input PWM signal. The wiring of the thruster and the controller can be found below. Further, the electric circuit for the BlueRobotic ESC can be found in a higher resolution in appendix A.

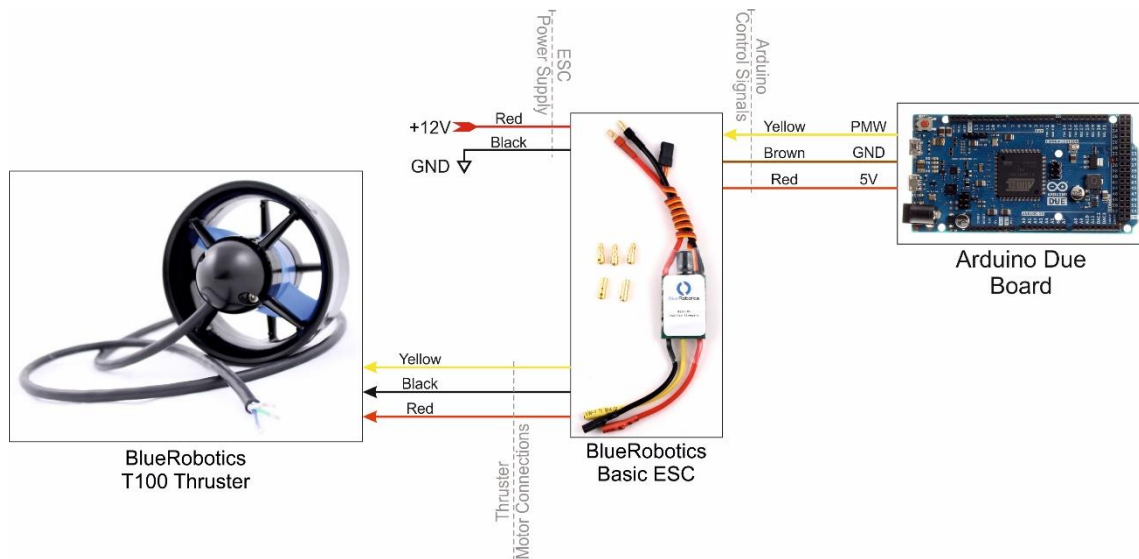


Figure 4-19: BlueRobotics T100 Thruster with ESC Motor Controller – Wiring Diagram

The input signal of the controller is a PWM signal with a max. update frequency of 400Hz, whereby the duty cycle of 1,100 μ s means full thrust backward, 1,500 μ s means stop and 1,900 μ s means full thrust forward. Further, the ESC need a 5V logic power supply and a GND, which both have to be connected with the Arduino board. The power supply is a 12V voltage. The ESC motor connections are connected with the T100 thruster.

The PWM signal generation in Arduino is realized by the Servo library [32]. The library is able to generate PWM signals on 12 output pins simultaneously with an update frequency of 50Hz. To increase the update rate of the thruster the Servos library is modified, as shown below.

```
1 // Minimum Time to Refresh Servos in Microseconds
2 #define REFRESH_INTERVAL 5000
```

Listing 4-12: Arduino Library – Servo.h Modification

The refresh interval in the library is changed to 500 μ s, which corresponds to a update frequency of 200Hz.

Because the Arduino DueTimer library is used in a previous step and both libraries are using the same timer, it is important to uncommment following line in the header of the DueTimer library. This enables the compatibility with the Arduino Servo library.

```
1 #define USING_SERVO_LIB true
```

Listing 4-13: Arduino DueTimer Library – Servo Library Compatibility

Finally, both libraries can be used simultaneously.

4.7 Inertial Measurement Unit

The Adafruit BNO055 Absolute Orientation Sensor [33] is a 9-DOF IMU used to calculate the orientation of the SUUV. The Inertial Measurement Unit (IMU) uses the Bosch BNO055 chip with a MEMS accelerometer, magnetometer and gyroscope. The chips has an integrated ARM Cortex-M0 based processor, which performs the sensor fusion

4. Electro-Mechanical Construction

and filtering on its own. The maximal update frequency is 100Hz for the absolute orientation. The IMU delivers either the calculated absolute orientation or the raw data in quaternions, Euler angles or vector form, Figure 4-20.

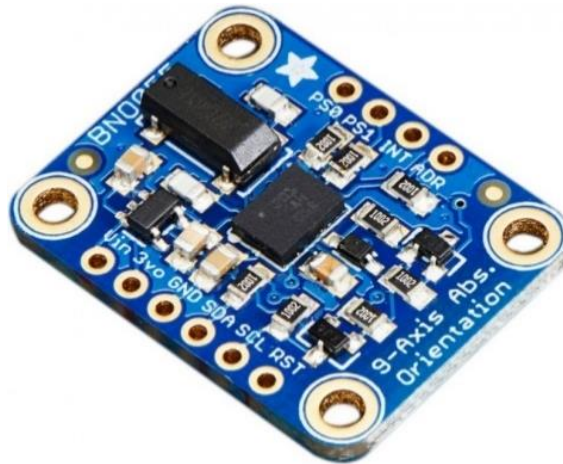


Figure 4-20: Adafruit BNO055 Absolute Orientation Sensor

The IMU communicates with the Arduino board via the I2C communication protocol. *Vin* and *GND* have to be connected with a voltage level between 3 – 5V. The *SCL* pin has to be connected with the I2C clock at the Arduino board, which is Pin 21 at the Arduino Due. The I2C data wire *SDA* has to be connected with the I2C data pin, which is Pin 20 on the Arduino Due.

Adafruit provides the driver and the libraries for the absolute orientation sensor, which can be found in the documentation of [33]. However, the following code includes the needed libraries for the Adafruit IMU. The library is integrated in an internal Arduino class and can be seen below.

```
1 // Adafruit BNO055 Sensor Libraries
2 #include <Adafruit_Sensor.h>
3 #include <Adafruit_BNO055.h>
4 #include <utility/imuMaths.h>
```

Listing 4-14: Adafruit IMU – Include Libraries

The *bno*-object enables the communication with the Adafruit IMU. The initializations and the creation of the “Adafruit_BNO055” object have to be done in the setup of the Arduino, which can be seen below.

```
1 Adafruit BNO055 AdafruitBNO = Adafruit BNO055(55);
2
3 void setup()
4 {
5     // ----- Initialize Adafruit BNO055 IMU -----
6     if (!IMU.AdafruitBNO.begin()) {
7         IMU.noBNO055 = true;
8         Serial.println("No BNO055 detected ... Check your wiring! \t");
9     }
10    else {
11        IMU.AdafruitBNO.setExtCrystalUse(true);
12        Serial.println("IMU Initialized! \t");
13    }
14 }
```

4. Electro-Mechanical Construction

Listing 4-15: Adafruit IMU – Setup Procedure

To get the absolute orientation of the IMU the sensor event has to be called, which returns the actual orientation of the IMU. Because the maximal update frequency of the sensor is 100Hz, a timer is implemented to update the IMU values frequently. This timer toggles a variable, which is checked in the main loop and by the “readBNO055”-function of the internal class. The code for the class function can be found below.

```
1 // ----- Read BNO055 IMU Values -----
2 void readBNO055 ()
3 {
4     // ----- Read IMU BNO055 Data -----
5     // Check if Sensor is Active and IMU Toggle is True
6     if (!noBNO055 && this->toggleIMU) {
7         // ----- Read EULER Angles Adafruit BNO055 IMU Data -----
8         this->AdafruitBNO.getEvent(&this->event);
9
10        phi = event.orientation.y;
11        theta = event.orientation.z;
12        psi = event.orientation.x;
13
14        // ----- Read RAW Adafruit BNO055 IMU Data -----
15        quat = this->AdafruitBNO.getQuat();
16
17        Quat_W = quat.w();
18        Quat_X = quat.x();
19        Quat_Y = quat.y();
20        Quat_Z = quat.z();
21
22        // Toggle IMU Read Trigger False Until Next Call
23        this->toggleIMU = false;
24    }
25 }
```

Listing 4-16: Adafruit IMU – Read Absolute and Raw Orientation

The raw data of the sensor in vector format can be calculated by the “getVector” or the quartanians by the “getQuat” statement. A more detailed description can be found on the Adafruit webpage.

4.8 Arduino / Arduino – Communication

Several task were introduced in the previous chapter, which are all handled by one Arduino board. A total of eight ISRs have to be handled, two pulse generators for the stepper driver, two encoder counter, one I2C communication for the IMU and one SPI communication for the SPI communication with the Ethernet board. All this tasks together bring the Arduino Due board on its limit, whereby it cannot be proven anymore that all task are handled in a time sufficient way. Therefore, as already mentioned in chapter 4.4.2, tasks were outsourced. For example, the count of the encoder ticks to a special encoder shield. In a further step, different tasks shall be distributed to two separate Arduino boards, which decreases the amount of work per Arduino.

4.8.1 Arduino Task Distribution

As already mentioned, a second Arduino shall be introduced to the project. Therefore, a smaller and less powerful Arduino Uno can be used. The main Arduino Due shall act as a master Arduino and handle all the important tasks as the Ethernet communication, generation of the stepper driver signal or the reading of the IMU. The handling of the encoder shield is shifted to the Arduino Uno. The Arduino Uno is based on a 5V logic level, as the encoder shield. Which means the tasks are distributed to two Arduinos based on the task logic level. Figure 4-21 below shows the distribution of the tasks and the communication between both boards.

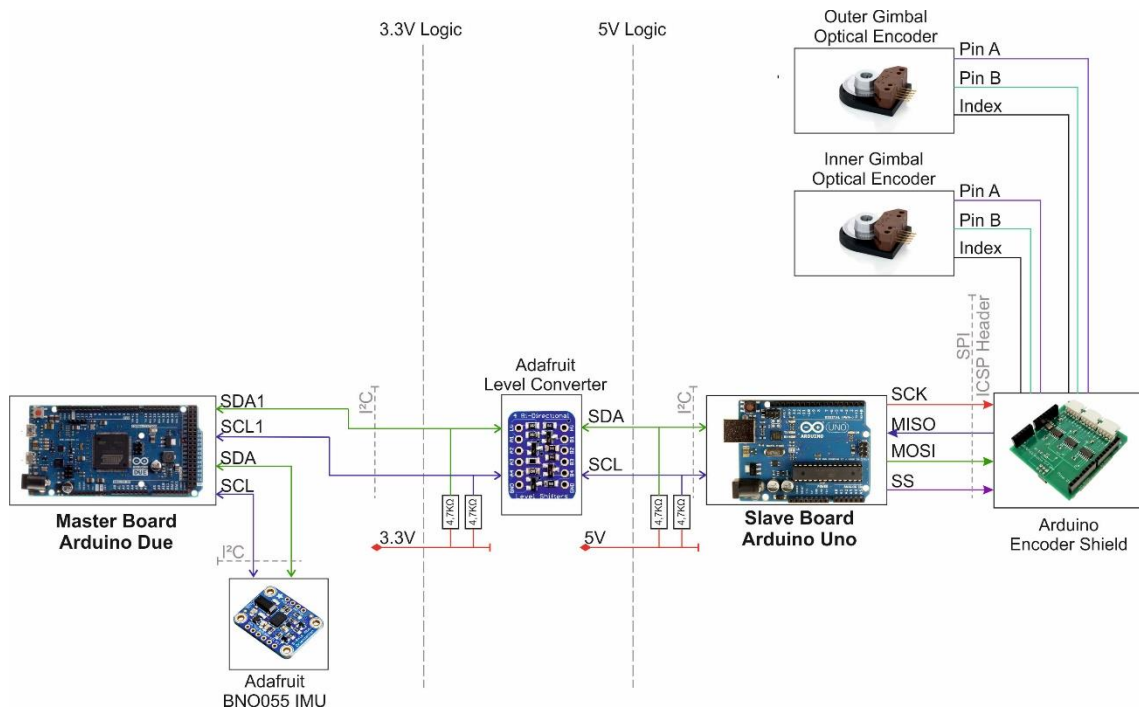


Figure 4-21: Schematic – Arduino/Arduino Communication

The communication between both Arduinos is realized via the I2C communication, which is a serial bus. The bus communicates via two wires; the data line (SDA) and the clock line (SCL). As a result of the usage of the Arduino Due Board, which operates at a voltage level of 3.3V, a voltage level converter board is needed. Therefore, the Adafruit 4-channel I2C-safe Bi-directional Logic Level Converter Board [34] is used. It can shift voltage levels bi-directional and it is used to shift clock and the data line of the I2C communication. It is important to use a I2C safe level shifter, because not every device provides this ability.

4.8.2 Software Implementation

This chapter describes the software implementation of the Arduino code. The Arduino Wire library [35] is used to handle the I2C communication. It provides several commands

4. Electro-Mechanical Construction

for the read, write and an event handler, which reacts on incoming data bytes. Such an event handler is implemented in the following chapter on the master and slave side.

4.8.2.1 Master Arduino

The Arduino Due has two hardware based I2C communication sockets. One is standardly used by the IMU library to exchange data. The Arduino joins the I2C bus as a master device on the second socket by the “wire1”-command, which is on pin SCL1 and SDA1. It is important to note that the SCL1 and SDA1 pins have no internal pull-up resistors. Therefore, external resistors are needed with a recommended value between $4.7k\Omega$ and $10k\Omega$. In this case, $4.7k\Omega$ resistors are used. The pull-up resistors are connected as shown in Figure 4-21. The second communication socket can be used by the following command in the setup.

```
1 // ----- Setup I2C Connection -----  
2 Wire1.begin(); //Join the bus as master on SCL1 and SDA1 Pins
```

Listing 4-17: Master Arduino – Wire1 Initialization

As mentioned, an event handler is implemented. Means, the master requests from the slave data and the slave is providing the requested data. This has the advantage that the I2C communication is just busy when actual data are needed. The code is created in an Arduino class, which contains the function for the request command. The values sent by the slave contain the two gimbal angles separated by a semicolon. On master side, this request is implemented as following:

```
1 void readData()  
2 {  
3     // Empty Char Array Buffers  
4     this->I2C_Buffer1[0] = '0';  
5     this->I2C_Buffer2[0] = '0';  
6     seperatorTrigger = false;  
7  
8     // Stop Active Wire Communication  
9     Wire1.endTransmission();  
10  
11     // Request Data from the I2C Slave  
12     Wire1.requestFrom(Slave_Address, BufferSize);  
13  
14     // Receive Data from I2C Slave  
15     // Slave may Send Less than Requested  
16     if (Wire1.available() == BufferSize) {  
17  
18         // Loop through Buffer Size  
19         for (int i = 0; i < BufferSize; i++) {  
20             // Read each Char from I2C  
21             recChar = Wire1.read(); // receive a byte as character  
22  
23             // Check for Char Seperator -> Set trigger  
24             if (recChar == ';') {  
25                 seperatorTrigger = true;  
26                 seperatorPos = i + 1;  
27             }  
28             else {  
29                 // Write Data to the Dependent Buffer  
30                 if (!seperatorTrigger) {  
31                     // Write I2C Value to Buffer 1
```


4. Electro-Mechanical Construction

```
32     this->I2C_Buffer1[i] = recChar;
33     }
34     else {
35         // Write I2C Value to Buffer 2
36         this->I2C_Buffer2[i - seperatorPos] = recChar;
37     }
38     }
39     }
40
41     // Convert Char Array to Float if Value is Not Out of Range
42     if (atof(this->I2C_Buffer1) <= 360 && atof(this->I2C_Buffer1) > 0.05) {
43         encoder1.Angle = atof(this->I2C_Buffer1);
44     }
45
46     if (atof(this->I2C_Buffer2) <= 360 && atof(this->I2C_Buffer2) > 0.05) {
47         encoder2.Angle = atof(this->I2C_Buffer2);
48     }
49     }
50 }
```

Listing 4-18: Master Arduino – Request Data from Slave

The slave address is the I2C address chosen by the slave. The buffer size depends on the amount of requested data. When wire1 is available with the right buffer size, means no data got lost during transfer or other error accurse, the function starts to read the data byte by byte. Each byte is checked if it is the semicolon separator and dependent of that, the byte is written in the buffer one or buffer two. At the end, both buffers are checked if they are in a reasonable range as expected, meaning from 0° to 360°. This is also a final check if a transfer error was made. Finally, the byte arrays are converted to a float by the “atof”-command.

4.8.2.2 Slave Arduino

On slave side the Arduino Uno is used, which has just one hardware based I2C socket. Therefore, the bus is joined with the “wire”-command and the code for the setup looks like.

```
1 // Startup I2C Communication
2 Wire.begin(Slave_Address); // Join I2C Bus with Address #42
3 Wire.onRequest(requestEvent_I2C); // Register Event
```

Listing 4-19: Slave Arduino – Wire Initialization

The bus is joined as slave with the slave address defined before. Further, an “onRequest”-event is generated with a specific function name. This function contains the write to I2C command, which expects a byte buffer. This buffer can be generated by the following code.

```
1 // ----- Generate I2C Buffer String -----
2 // Conversion from Float to Char-String (Not Supported for Uno in sprintf)
3 dtostrf(encoder1.Angle, 5, 2, strEncoder1_Angle);
4 dtostrf(encoder2.Angle, 5, 2, strEncoder2_Angle);
5 sprintf(I2C_Buffer, "%s;%s", strEncoder1_Angle, strEncoder2_Angle);
```

Listing 4-20: Slave Arduino – Buffer Generation

The “onRequest”-function finally writes the buffer to the I2C bus if data are requested by the master. This is an extreme efficient way to communicate between two devices

because no unnecessary data are transferred. The “onRequest”-function can be seen below.

1
2
3
4
5
6

```
// ----- I2C Communication Event Handler -----  
void requestEvent_I2C()  
{  
  // Write I2C Buffer to Connection  
  Wire.write(I2C_Buffer);           // Respond with Message of 6 Bytes  
}
```

Listing 4-21: Slave Arduino – onRequest Function

The communication between both devices is very stable. It is important that certain safety checks are implemented in case the bytes are not transferred in a right matter.

4.9 MATLAB / Arduino – Communication

As a final step, the orientation of the SUUV shall be controlled. Because of the non-linearity of the dynamics of the system, such control algorithms are very expensive and need a high computational effort. However, regarding the systems flexibility it is easier to program such control algorithms in MATLAB Simulink than coding them directly on the Arduino. Therefore, a proper way to communicate between the Arduino board and MATLAB Simulink has to be found.

4.9.1 Serial Communication

One of the easiest solutions is the Serial interface of the Arduino board. It can communicate via USB directly with MATLAB Simulink. Additionally, the system is very cheap because no additional devices are needed. One possibility to code the Serial communicate between Arduino and MATLAB Simulink is the MATLAB Simulink Support Package for Arduino [36]. The package is provided by MATLAB and enables to read and write directly I/Os of the Arduino board. However, the strict programming and the low flexibility of the package is not sufficient for the project. Therefore, a direct implementation of the Serial communication as a Simulink function is done. A Level-2 MATLAB S-Functions is programmed, which allows to create an individual block in Simulink. The whole implementation of the function is shown in the chapter below.

After implementing the code, several drawbacks have been found: The Serial port is always occupied and cannot used by the Serial Monitor of Arduino; The Serial communication is pretty slow for huge data amounts, which makes it necessary to increase the baud rate to the upper limit. This leads to the problem that the maximal cable length is limited to 1m, which is not acceptable for further water tests of the SUUV. Therefore, it is decided to replace the Serial communication with an Ethernet communication.

4.9.2 Ethernet Communication

The Sparkfun Arduino Ethernet Shield 2 [37] (Figure 4-22) provides the possibility to connect the Arduino board with any Ethernet capable device or the internet and communicate via either TCP or UDP.

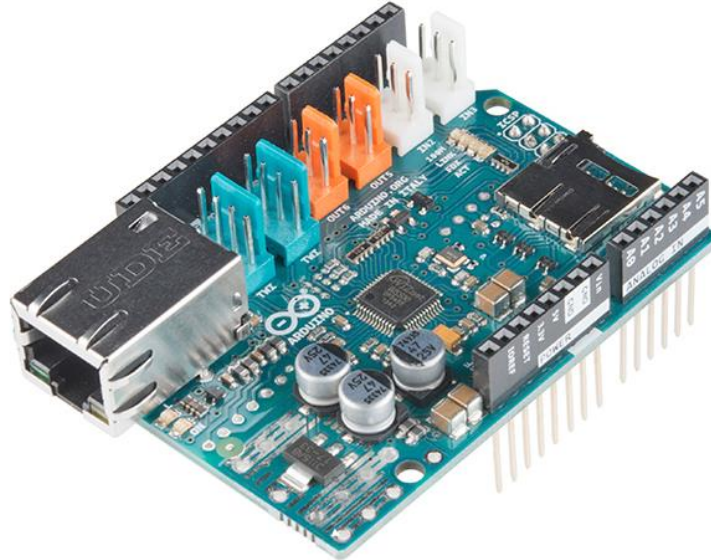


Figure 4-22: Sparkfun Arduino Ethernet Shield 2

The Ethernet shield communicates with the Arduino via the SPI interface, through the ICSP header. Additionally, the shield provides a standard RJ45 Ethernet jack for the communication with any Ethernet device. The figure below shows a schematic of the connections between the devices.

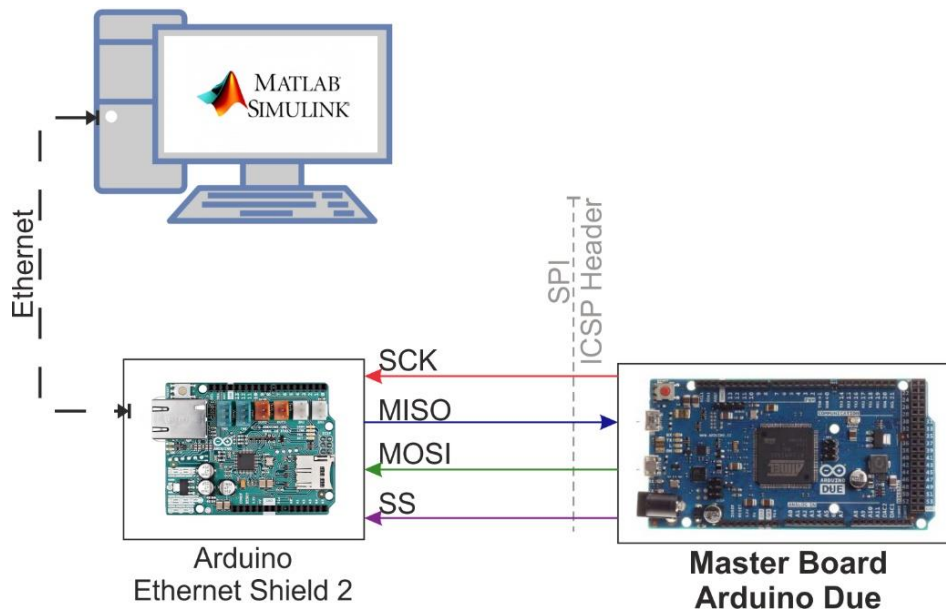


Figure 4-23: Schematic – Arduino MATLAB Communication

The Simulink function created for the Serial communication is adapted for the Ethernet communication. The basic algorithm behind is explained in the chapter below.

4. Electro-Mechanical Construction

4.9.2.1 Basic Algorithm

The algorithm for the communication is based on a cycling read and write loop. This means, at every moment one device is sending and the other one is reading data. The MATLAB code sends a string with all data for Arduino. The Arduino board is listening and as soon as Arduino receives the data, Arduino is sending the own data string. The MATLAB is waiting until it receives the Arduino data string. After that, the cycle is repeated. This avoids that each device hears itself. The MATLAB command string looks like:

```
1 ['M:' BLDC_Enable ';' BLDC_RPM ';' stepper1_RPM ';' stepper2_RPM ';'
2 BLDC_Thruster1_Throttle ';' BLDC_Thruster2_Throttle ';']
```

Listing 4-22: MATLAB Ethernet Communication – Command String

The 'M:' is an additional indicator for Arduino that the string is coming from MATLAB. Each value is separated by a semicolon. On the other hand, the Arduino command string looks like:

```
1 ['A:' encoder1.Angle ';' encoder2.Angle ';' IMU.phi ';' IMU.theta ';' IMU.psi
2 ';' IMU.Quat_W ';' IMU.Quat_X ';' IMU.Quat_Y ';' IMU.Quat_Z ';']
```

Listing 4-23: Arduino Ethernet Communication – Command String

The 'A:' indicates that the string is coming from Arduino.

4.9.2.2 MATLAB Implementation

As mentioned before, a Level-2 MATLAB S-Functions is used for the serial communication. This chapter explains the adaption and the schematic of the code for the Ethernet communication in MATLAB Simulink. A Level-2 MATLAB S-Functions consists of several smaller functions.

```
1 function Arduino_Board_Ethernet(block)
2
3 setup(block);
```

Listing 4-24: MATLAB Ethernet Communication – Block Generation

```
1 function setup(block)
2
3     %% Register Number of Input and Output Ports
4     block.NumInputPorts = 1;
5     block.NumOutputPorts = 1;
6
7     %% Setup Functional Port Properties to Dynamically Inherited
8     block.SetPreCompOutPortInfoToDynamic;
9
10    %% Setup Functional Port to Default
11    block.SetPreCompPortInfoToDefaults;
12
13    %% Setup Input Ports
14    block.InputPort(1).Dimensions = 1;
15    block.InputPort(1).DirectFeedthrough = false;
16    block.InputPort(1).DatatypeID = 8; % boolean
17    block.InputPort(1).Complexity = 'Real';
18    block.InputPort(1).SamplingMode = 'Sample';
19
```

4. Electro-Mechanical Construction

```
20     %% Setup Output Ports
21     block.OutputPort(1).Dimensions = 2;
22     block.OutputPort(1).DatatypeID = 0; % double
23     block.OutputPort(1).Complexity = 'Real';
24
25     %% Set Block Sample Time to 100Hz
26     block.SampleTimes = [0.01 0];
27
28     %% Set the Block simStateCompliance to Default
29     block.SimStateCompliance = 'DefaultSimState';
30
31     %% Register Methods
32     block.RegBlockMethod('InitializeConditions', @InitArduinoEthernet);
33     block.RegBlockMethod('Terminate', @TerminateArduino);
34     block.RegBlockMethod('Update', @Output);
```

Listing 4-25: MATLAB Ethernet Communication – Setup

The register block method allows attaching certain other function to the block with specific properties. The first function calls the “setup”-function for the block. The number of the in- and outputs have to be defined and the certain properties for the I/Os have to be set. Line 38 in the code above calls the “InitArduinoEthernet”-function, which is called just once during the initialization of the block. The code below shows the initialization of the Ethernet communication.

```
1  function InitArduinoEthernet(block)
2
3     % Global variable
4     global myArduinoUDP;
5
6     %% Create Arduino Ethernet Connection Object
7     myArduinoUDP = udp('192.168.1.110', 'RemotePort', 8888, 'LocalPort', 8080);
8     myArduinoUDP.Timeout = 0.01;
9
10    %% Open Arduino Serial Connection
11    fopen(myArduinoUDP)
```

Listing 4-26: MATLAB Ethernet Communication – Initialization

The “myArduinoUDP”-object has to be set to global, because it is used in other functions too. Afterwards the object is created. The IP-address has to be set in the space of the network card of your computer. In this case to '192.168.1.xxx'. The last digits can be chosen anyhow. For the remote port and the local port, a port number without any special purpose should be chosen. Afterwards the connection can be opened. It is important to do these steps in the initialization block, otherwise the communication would be opened at each time step. The timeout has to be set to the sampling time, otherwise the communication freezes when communication problems appear.

A block with the ‘update’ property is created. Therefore, the function is called at each time step. The code can be seen below.

```
1  function Output(block)
2     % Global variable
3     global myArduinoUDP;
4
5     try
6         %% Send to Arduino
7         % Read Input Values from Block
8         stepper1_RPM = num2str(block.InputPort(1).Data(1), '%3.2f');
9         stepper2_RPM = num2str(block.InputPort(1).Data(2), '%3.2f');
```

4. Electro-Mechanical Construction

```
10
11     % Create Send Command
12     arduinoSendData = ['M:' stepper1_RPM ';' stepper2_RPM ';']
13
14     % Send Command-String to Arduion
15     fwrite(myArduinoUDP, arduinoSendData)
16
17
18     %% Read from Arduino
19     % Read response from Arduino
20     arduinoReceivedData = fgetl(myArduinoUDP)
21
22     % Check if Received Command-String is Valid
23     % String Not Empty && Identification is Existing
24     if ~isempty(arduinoReceivedData) && strcmp(arduinoReceivedData(1:2), ...
25         'A:')
26
27         % Split String into Substrings
28         tmpArduinoReceivedData = strsplit(arduinoReceivedData(3:length ...
29             (arduinoReceivedData)), ';');
30
31         % Save Data to Local Variables
32         encoder1_RPM = str2double(tmpArduinoReceivedData(1));
33         encoder2_RPM = str2double(tmpArduinoReceivedData(2));
34
35         % Write Data do Outputs
36         block.OutputPort(1).Data = [encoder1_RPM encoder2_RPM];
37
38     end
39 catch exception
40
41     getReport(exception, 'extended', 'hyperlinks', 'ON')
42
43 end
```

Listing 4-27: MATLAB Ethernet Communication – Update

The “fwrite”-command allows to write a string to the Ethernet object, which writes the string to the open port. The “fgetl”-command reads from the Ethernet port. The string, which is sent do Arduino is prepared in the steps before, as described in chapter 4.9.2.1. The Arduino command string is split in the separate values afterwards. Finally, the terminate block writes zero values to all Arduino variables to stop all motors, closes the connection and delete all Ethernet objects, which can be seen below.

```
1 function TerminateArduino(block)
2
3     global myArduinoUDP;
4
5     arduinoSendData = ['M:' '0' ';' '0' ';' '0' ';' '0' ';' '0' ';' '0'];
6
7     % Send Command String to Arduino
8     fwrite(myArduinoUDP, arduinoSendData)
9
10    % Release all Communication Objects
11    delete(instrfindall)
```

Listing 4-28: MATLAB Ethernet Communication – Terminate

4.9.2.3 Arduino Implementation

On Arduino side, several parameters and libraries are needed. The Sparkfun Arduino Ethernet Shield 2 [37] has the Wiznet W5500 Ethernet chip, which needs the <Ethernet2.h> and the <EthernetUdp2.h> libraries. The needed parameters are: the

4. Electro-Mechanical Construction

MAC address of the Ethernet shield, which is given on the backside of the shield; the IP address, which is defined by the network space of the network card, but do not use the same address as used on MATLAB side; and the local port used by MATLAB. The parameters can be seen below:

```
1 #include <SPI.h> // SPI Library
2 #include <Ethernet2.h> // Ethernet Library
3 #include <EthernetUdp2.h> // Ethernet UDP Library
4
5 byte mac[] = {0x90, 0xA2, 0xDA, 0x10, 0x6E, 0x8F}; // Mac Adresse
6
7 IPAddress ip(192, 168, 1, 110);
8 const unsigned int localPort = 8888; // Local Port to Listen On
```

Listing 4-29: Arduino Ethernet Connection – Ethernet Objects

Additional parameters for the puffer size are needed. It is important here that the puffer size should be selected static, because a dynamic puffer size slows down the CPU at high UDP read and write cycles. The size of the puffer is dependent of the amount of data, which is transferred. The code can be found below:

```
1 // EthernetUdp Instance Enables to Send and Receive Packets via UDP
2 EthernetUDP Udp;
3
4 // Buffer to Hold Incoming Packet
5 char packetBuffer[UDP_TX_PACKET_MAX_SIZE];
6
7 // Define MATLAB Command Strings
8 String MatlabRawData_Received;
9 char MatlabRawData_Send[64] = "";
10 String tmpMatlabData = "";
11 int packetSize = 0;
```

Listing 4-30: Arduino Ethernet Connection – Ethernet Variables

In the Arduino setup, the Ethernet communication is created. The UDP timeout time should be set to the same vale as the chosen sample time, in this case $100Hz$ and therefore a timeout of $10ms$, which can be seen below:

```
1 void setup() {
2
3 // ----- Setup Ethernet UDP Connection -----
4 Ethernet.begin(mac, ip);
5 Udp.begin(localPort);
6 Udp.setTimeout(10);
7 }
```

Listing 4-31: Arduino Ethernet Connection – Setup Ethernet Communication

As described in chapter 4.9.2.1, the Arduino board waits until a MATLAB command string is received and then returns the Arduino command string. Therefore, a function is created which is called in the main loop of the program. The Arduino program check with the command “Udp.parsePacket(;)” if a package is received. Afterwards in line 5 the package is read from the UDP connection. If in line 7 the MATLAB command string is detected, the string is split by the function “getStringPartByNr”, which is explained later. Afterwards the Arduino command string is returned to the remote IP address and the remote port. The whole code can be found below:

```
1 // ----- Ethernet MATLAB Communication -----
```


4. Electro-Mechanical Construction

```
2 void udpMATLABCommunication ()
3 {
4   // Check if a Ethernet Command is Available
5   packetSize = Udp.parsePacket();
6
7   if (packetSize) {
8     // Read Data from Ethernet Connection
9     Udp.read(packetBuffer, packetSize);
10
11     MatlabRawData_Received = packetBuffer;
12
13     if (MatlabRawData_Received.substring(0, 2) == "M:") {
14       // ----- Read Data from MATLAB -----
15       // Read Motor Values from Raw Data String
16       tmpMatlabData = MatlabRawData_Received.substring(2,
17         MatlabRawData_Received.length());
18
19       // Read Particular Values from Raw Data
20       BLDC_Gyro.Enable = getStringPartByNr(tmpMatlabData, ';',
21         0).toFloat();
22       BLDC_Gyro.RPM_set = getStringPartByNr(tmpMatlabData, ';',
23         1).toFloat();
24       stepper1.RPM_set = getStringPartByNr(tmpMatlabData, ';',
25         2).toFloat();
26       stepper2.RPM_set = getStringPartByNr(tmpMatlabData, ';',
27         3).toFloat();
28       BLDC_Thruster1_Set = getStringPartByNr(tmpMatlabData, ';',
29         4).toFloat();
30       BLDC_Thruster2_Set = getStringPartByNr(tmpMatlabData, ';',
31         5).toFloat();
32
33
34       // ----- Write Data to MATLAB -----
35       sprintf(MatlabRawData_Send,
36         "A:%.2f;%.2f;%.3f;%.3f;%.3f;%.3f;%.3f;%.3f;%.3f", encoder1.Angle,
37         encoder2.Angle, IMU.phi, IMU.theta, IMU.psi, IMU.Quat_W, IMU.Quat_X,
38         IMU.Quat_Y, IMU.Quat_Z);
39
40       // Reply to the Remote IP Address and Port
41       Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
42       Udp.write(MatlabRawData_Send);
43       Udp.endPacket();
44     }
45   }
46 }
```

Listing 4-32: Arduino Ethernet Connection – Update Ethernet

The “getStringPartByNr”-function returns a string part detected in the command string at a certain index position and separated by a certain separator. The whole code for the function can be found below:

```
1 // ----- Split String by Separator -----
2 String getStringPartByNr(String data, char separator, int index)
3 {
4   // Return the Part Nr Index
5   int stringData = 0; // Count Data Part Nr
6   String dataPart = ""; // Hold the Returned Text
7
8   // Go Through the Whole String to Find Separator Positions
9   for (int i = 0; i < data.length() - 1; i++) {
10
11     if (data[i] == separator) {
12       // Count the Number of Times Separator Character Appears
13       stringData++;
14     }
15     else if (stringData == index) {
16       dataPart.concat(data[i]);

```


4. Electro-Mechanical Construction

```
17 }
18 else if (stringData > index) {
19     // Return Text and Stop if the Next Separator Appears
20     return dataPart;
21     break;
22 }
23 }
24
25 // Return Text if this is the Last Part
26 return dataPart;
27 }
```

Listing 4-33: Arduino Ethernet Connection – Split Command String

The sample time of the Ethernet communication is defined with 100Hz, which fulfils all given requirements. Finally, the implementation of the Level-2 MATLAB S-Functions in MATLAB Simulink looks like:

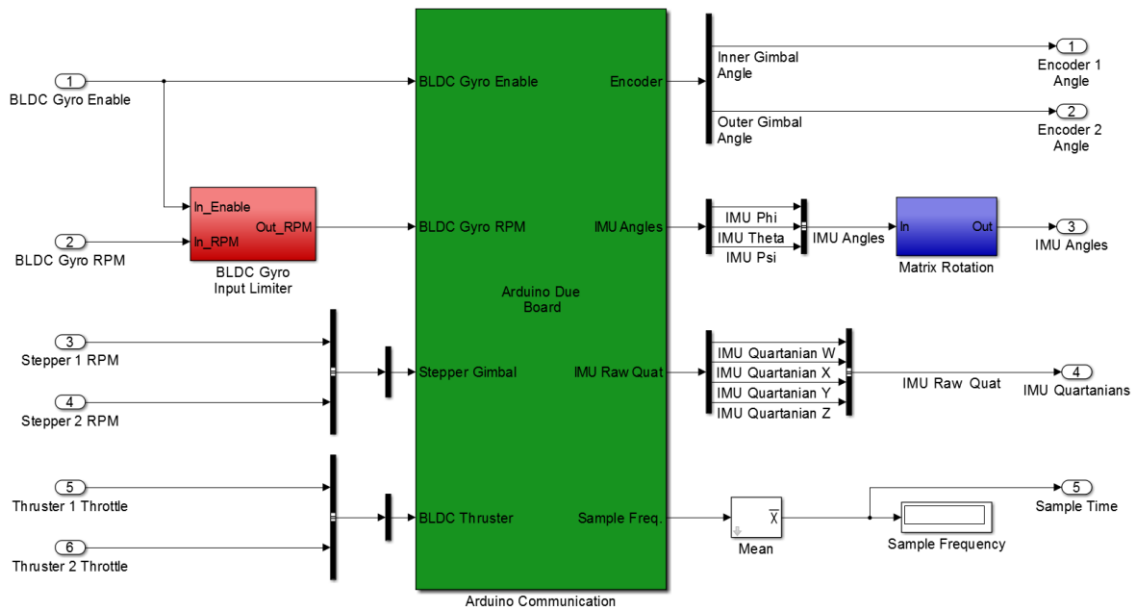


Figure 4-24: Arduino Ethernet Connection – MATLAB Function in Simulink

On the left side the different inputs can be seen. Whereby, the gyro speed is limited during startup for a smooth acceleration. On the right side, the actual encoder values and the orientation of the swimmer can be measured. The in- and outputs of the block are defined as multidimensional, thus multiplexer and demultiplexer are needed.

4.9.2.4 Setup the Windows Firewall

To admit the Ethernet communication on your Windows operating system, it is important to setup the Windows firewall. Both the remote and the local port have to be added to the input and output rules of your Windows system. It is important to enable to communication for the selected ports to allow the proper communication of the devices.

4.10 Software Task Distribution

This chapter deals with the interaction of the different software tasks. Several functions are distributed into software tasks on different Arduino Boards. The following Figure 4-25

4. Electro-Mechanical Construction

shows the two used Arduino boards and the Simulink control system. On the left side the MATLAB Simulink routine can be seen. In the middle, the Arduino Due tasks and on the right side the Arduino Uno are shown. Both containing the setup, the loop as the Interrupt Service Routines (ISR1 and ISR2), timer functions (timer1) and the event handler (event1). A more detailed picture of the software tasks distribution can be found in the appendix C.

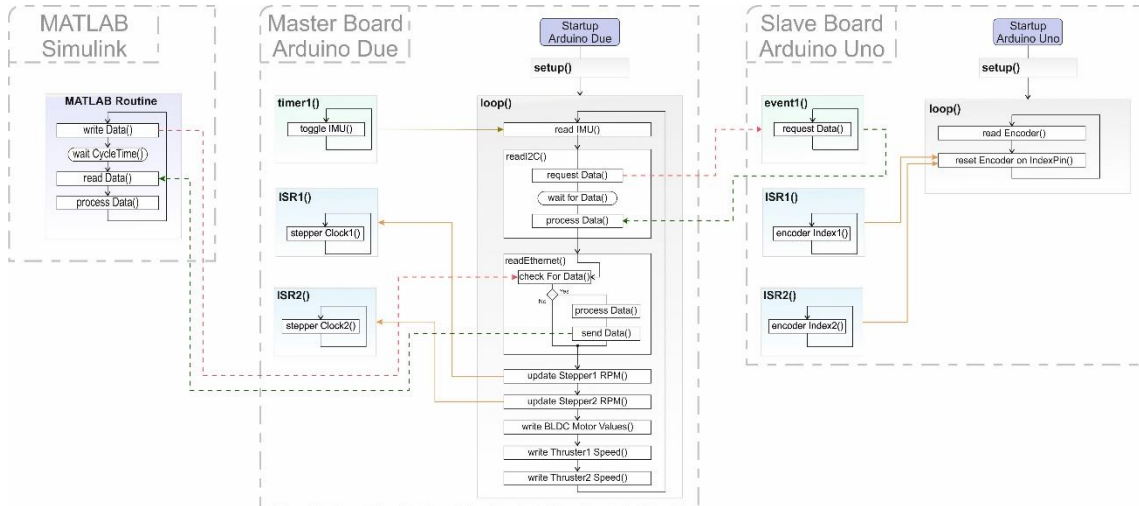


Figure 4-25: Flow Chart – Software Task Distribution

The MATLAB routine consists of firstly the write cycle, secondly the waiting for the Arduino answer and finally the processing of the received data cycle. The cycle runs with a sample frequency of 100Hz.

The Arduino Due is processing after the successful startup, the two ISRs, which are dealing with the generation of the pulses for the stepper driver. The ISRs get the new stepper speeds from the main loop. The timer function toggles the IMU read cycle in the main loop with a sample frequency of 100Hz. The “readI2C”-function is requesting data from the Arduino Uno event and waiting until it receives it. Finally, the data is processed. The “readEthernet”-function is checking for data from Simulink. If data is provided, it is processed and the actual Arduino values returned to Simulink. Finally, the Arduino Due is updating the motor velocities.

The Arduino Uno is initializing all the ISR and the event handler during startup. Afterwards, in each cycle the encoder values are read and if an interrupt appear on the encoder index pin, the encoder is reset. The event handler for the I2C communication works independent from the main loop and sends on request the encoder values to the Arduino Due board.

The interaction between the different devices can be seen by the dashed lines. The arrow indicates in which direction the command is going. The solid arrows between the ISR, timer functions or the event handler shows the interaction with the referring task.

4.11 Electrical Circuits

Finally, all the needed components to control the SUUV are combined. Two stepper motors rotate the DGCMG, whereby the pulses needed for the stepper driver are generated by the master Arduino Due board. The high speed center wheel is proposed by a BLDC motor, which is controlled by a speed controller. This speed controller gets the input signals from the master Arduino Due, too. For the linear motion of the swimmer, two brushless T100 thrusters are used. This thruster are controlled by BlueRobotics ESC, which get a PWM signal generated by the Arduino Due. An Adafruit BNO055 absolute orientation sensor is used to measure the position in global coordinates of the SUUV. Finally, the IMU communicates via I2C communication with the Arduino Due.

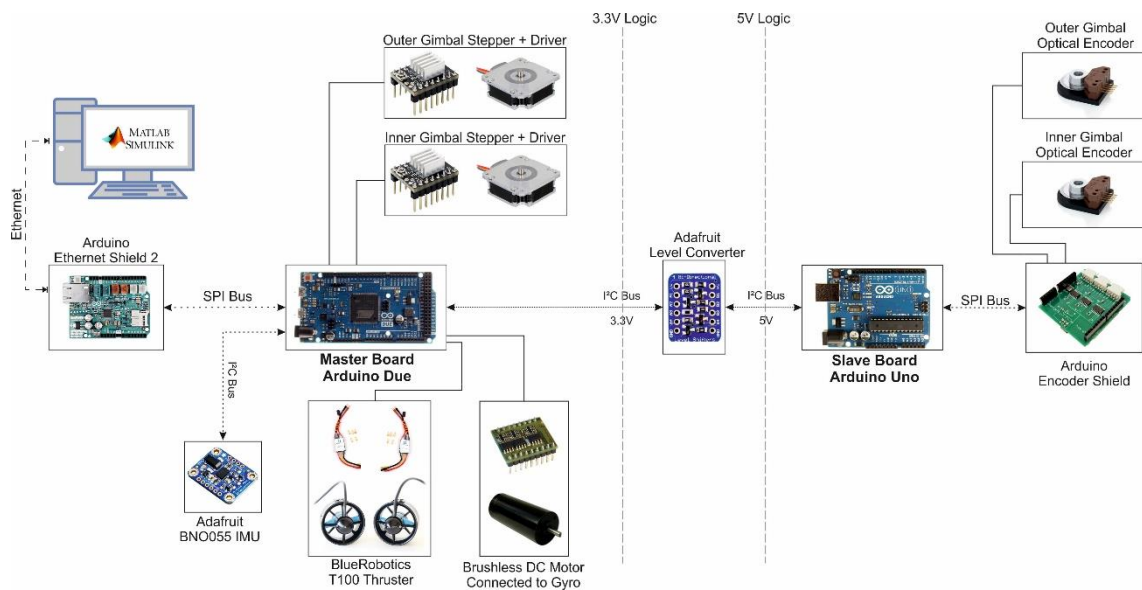


Figure 4-26: Electrical Circuit and Logic Connections of the SUUV Electronics

The orientation of both gimbals are measured by two optical encoders. This encoders work with a 5V logic level and the pulses generated by the encoder are counted by an Arduino encoder shield. This shield is moved to distribute the workload to a second slave Arduino Uno. The communication of both Arduinos is realized via I2C communication. Important is that a level shifter is needed because both Arduinos work on different logic levels. Finally, a system is needed to control the orientation of the robot. Because of the computational expensive non-linear control algorithms used, this work is done by MATLAB Simulink. The communication between Simulink and the Arduino is realized via the Arduino Ethernet Shield 2 which provides a RJ45 Ethernet jack for the communication with the computer and on the other hand side communicates via SPI interface with the Arduino.

4.11.1 Electrical Power Supply

The electrical circuit consists of several logic levels and devices. Different devices also need different power supply levels. The SUUV is supplied by 12V and 24V, as shown in the figure below.

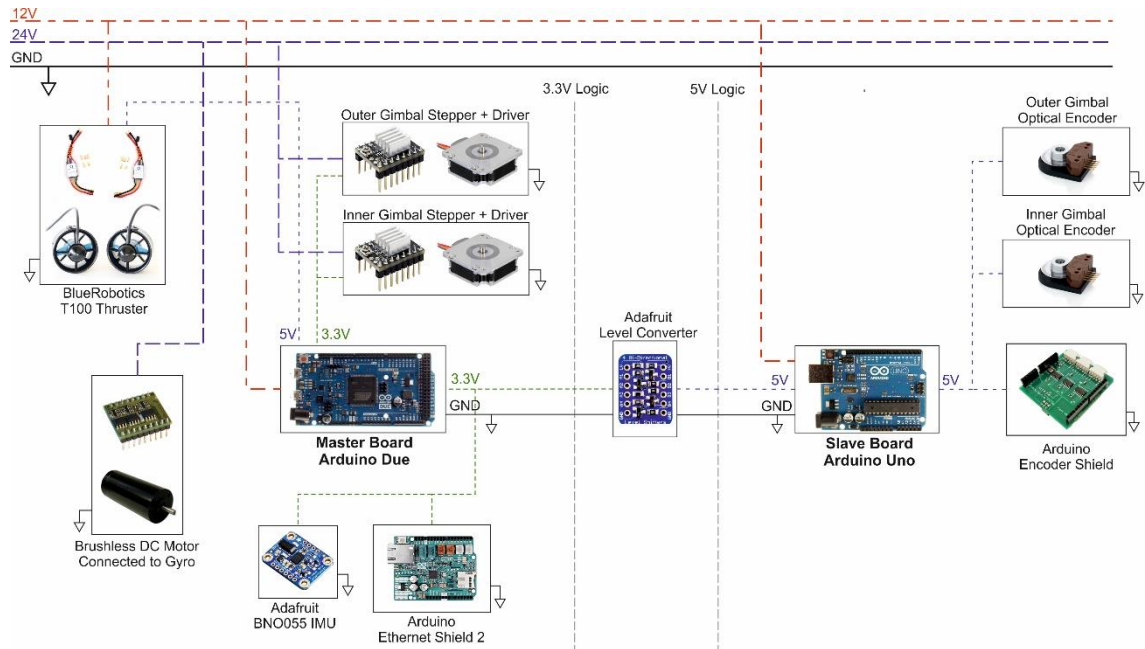


Figure 4-27: Power Supply of the SUUV Electronics

The 12V power supply is used for the Arduinos, the stepper motors and the thrusters. The stepper driver, the brushless DC motor and the driver of the center wheel is powered by 24V. The logic levels of 3.3V and 5V are generated by the Arduinos whereby both are connected with the logic level shifter and a common ground. The IMU, the stepper driver and the Ethernet shield are powered by 3.3V power level; the optical encoders, the encoder shield and the ESC for the thruster are powered by 5V power level. Each device is additionally connected to the common ground.

4.11.2 Logic Signals Connections

This chapter contains an overlook of the devices with its logic level and the signal connections to the regarding Arduino. From Figure 4-28 below it can be seen, that two logic levels of 3.3V and 5V are used.

4. Electro-Mechanical Construction

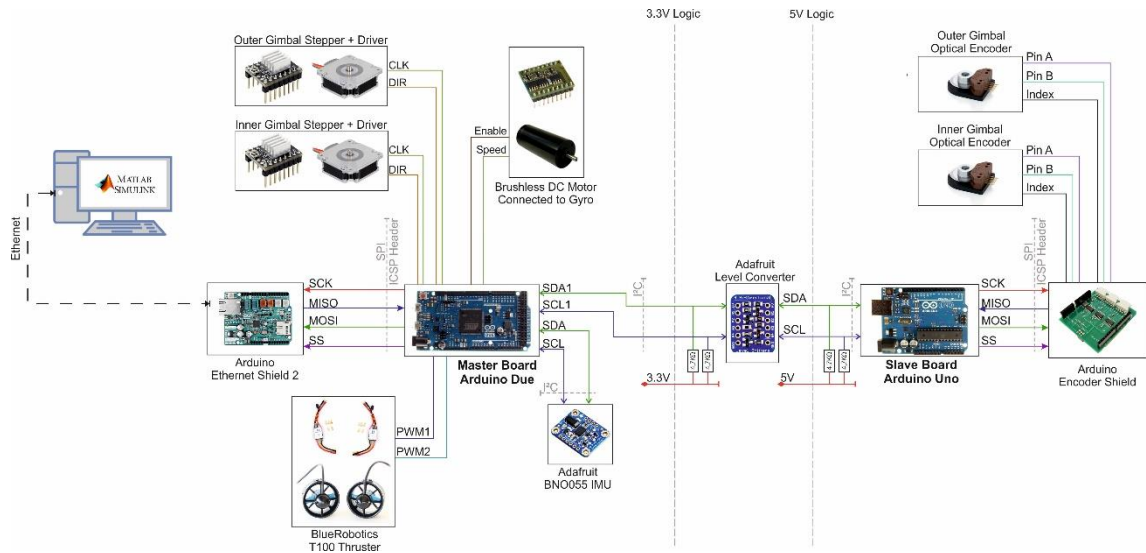


Figure 4-28: Logic Signal Connections of the SUUV Electronics

The SPI connections consists of four wires, the Serial Clock (SCK), the Master In Slave Out (MISO), the Master Out Slave In (MOSI) and the Slave Select (SS). The SS line is used to select the device for the communication. For example, the Ethernet Shield used Pin 10. The I2C communication just needs two wires, the data line (SDA) and the clock line (SCL).

Both optical encoders have a Pin A and Pin B for the pulses, which are counted by encoder shield. Additional, both devices have an index pin. All the pins are connected with the encoder shield. Each stepper driver need a pulse signal (CLK) and a direction signal (DIR) for the control of the motion of the stepper motor. The brushless DC motor driver for the gyro needs an enable signal and the analog speed command. The BlueRobotics T100 thruster are controlled with a PMW signal for each thruster ESC. The IMU communicates via I2C with the Arduino on the first I2C port of the Arduino Due. MATLAB Simulink and the Ethernet Shield are connected via an Ethernet cable.

This chapter described all the electrical circuits. Whereby, the electric circuit for the logic connections, the signal flow and the power supply for the devices are shown. All the electrical circuits can be found in a higher resolution in the appendix B.

5. Manufacturing

This chapter gives an overview about the different steps of the manufacturing process. As mentioned in chapter 2 the SUUV has been designed by Mohsen Saadat. The fabrication took place at the different machine shops at the University of California, Berkeley. Several students worked on the manufacturing of the parts. The final assembly and testing is part of a this project.

5.1 Fabrication and Design

The swimmer consists of three main part. The fuselage, which contains the double gimbal system and two end caps, where the thruster are mounted. The design introduced in chapter 2 is developed in a way that the body of the swimmer is natural buoyant. Means, the materials of the different parts varies that the robot is heavy enough to hold naturally its heave position and to be balanced in water. The size of the body shall be small, so that no additional dead mass is needed. The fuselage of the body consists of acrylic parts. The Double Gimbal Control Moment Gyro consists of aluminum and steel parts. The Fabrication of the parts is done by students which work in the laboratory. The parts are fabricated by a laser cutter and some of the acrylic parts are 3D printed. As mentioned, it is important that the swimmer is naturally balanced, which makes the fabrication process much more complex. The high speed center wheel is turned and might get balanced in a further step to avoid vibrations at high rotational speed.

5.2 Final Assembly

The final assembly and the wiring of the electrical components took place as a part of this project. The wiring is done based on the electrical design created in chapter 4.11. The front view of the assembled swimmer can be seen in the figure below.

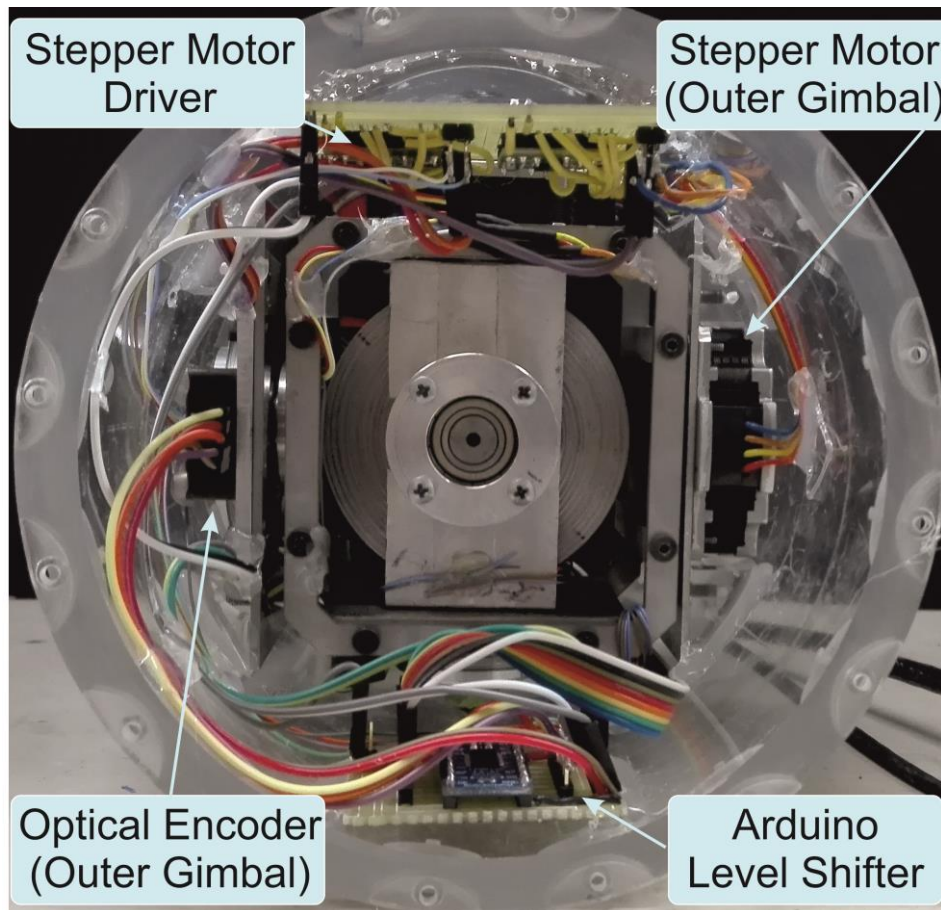


Figure 5-1: Front View of the Assembled Swimmer

The assembled double gimbal system can be seen in the center of the swimmer. The high speed gyro wheel is hold at its position by two bearings. The center wheel can rotate around two axis and spin at high speed. Because of the space limitation, the electronic is distributed over the whole body. The following picture shows the rear view of the assembled swimmer.

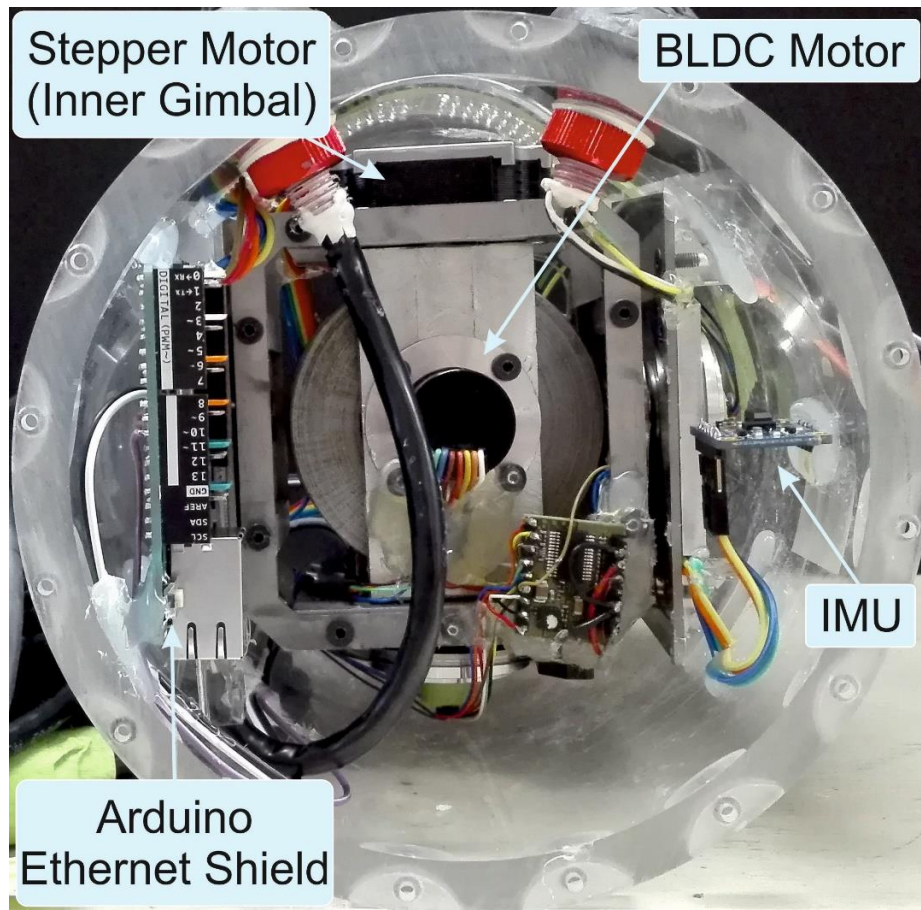


Figure 5-2: Rear View of the Assembled Swimmer

At the back, the Ethernet shield and the IMU can be seen. In the front the stepper driver and the level shifter is located. Note, that in this view because of a better overlook the Arduino Due, the Arduino Uno and the Encoder shield cannot be seen. They are located in the front of the swimmer. The ESC for the thruster are located in the back of the swimmer and cannot be seen at this picture neither. The power supply and the Ethernet cable are carry through two special underwater screws which are sealed with epoxy. The picture below shows the side view of the whole SUUV at its old design with the two thruster in the front and the back.

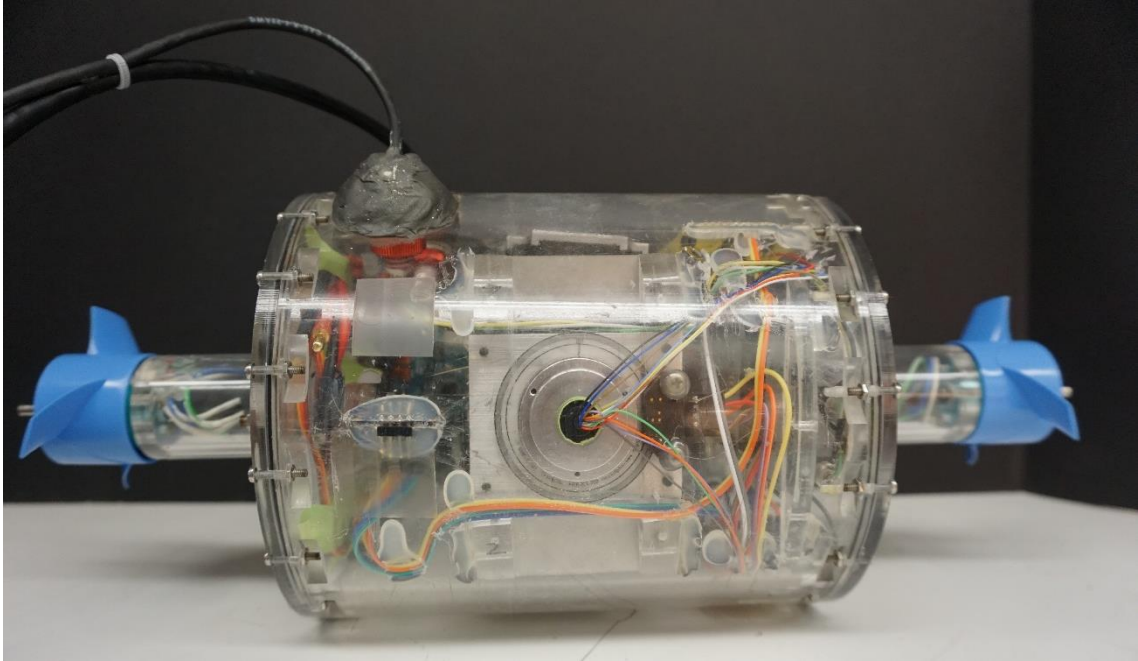


Figure 5-3: Side View of the Assembled Swimmer

The two thrusters are counter rotating and enable an additional DOF for the position control of the swimmer. At the top, the power supply cable and the Ethernet cable can be seen. Both cables carry all necessary signals for the robot. The design shown in this picture varies to the design introduced in the previous chapter because several design changes were made during the final assembly.

5.3 Waterproofness

An important part when building an undersea robot is the waterproofness of itself. This is given by the usage of acrylic parts for the fuselage of the swimmer. As mentioned before, the robot consists of three main parts, two end caps and the fuselage. The end caps are mounted to the body with 16 screws. O-rings are located between the end caps and the body. This seals the body against leakage. The power supply cable and the Ethernet cable are carried through special underwater screws, which are sealed with epoxy. For a final leakage test, the fuselage has been put for more than 24 hours in a water tank to proof its waterproofness.

6. Swimmer Agility Tests

After submerging the swimmer, it is observed that regarding practical issues and the instability of the swimmer in water an evaluation of the first agility results is very difficult. Therefore, it is decided to hang the robot in air. Different hanging methods can be used to eliminate the DOF of the swimmer in air. With these methods the performance of the DGCMG in the individual axis with the regarding gimbals can be evaluated. The following chapters show the different used hanging methods and the implementation of the one and two axis agility tests.

6.1 One Axis Agility Results

A one-axis agility test shall be performed with the swimmer in air. Therefore, two DOF of the swimmer have to be eliminated. Finally, the performance of the DGCMG for the yaw rotation shall be evaluated. For this test, the outer gimbal is fixed in vertical position and the inner gimbal rotates around the vertical y -axis. The following chapter shows the used hanging method and the performance of the DGCMG.

6.1.1 Hanging Method for One DOF

As mentioned, for the first performance tests of the swimmer two DOFs have to be eliminated. Therefore, the swimmer is hanged with a rope to the top of the room, shown in Figure 6-1.

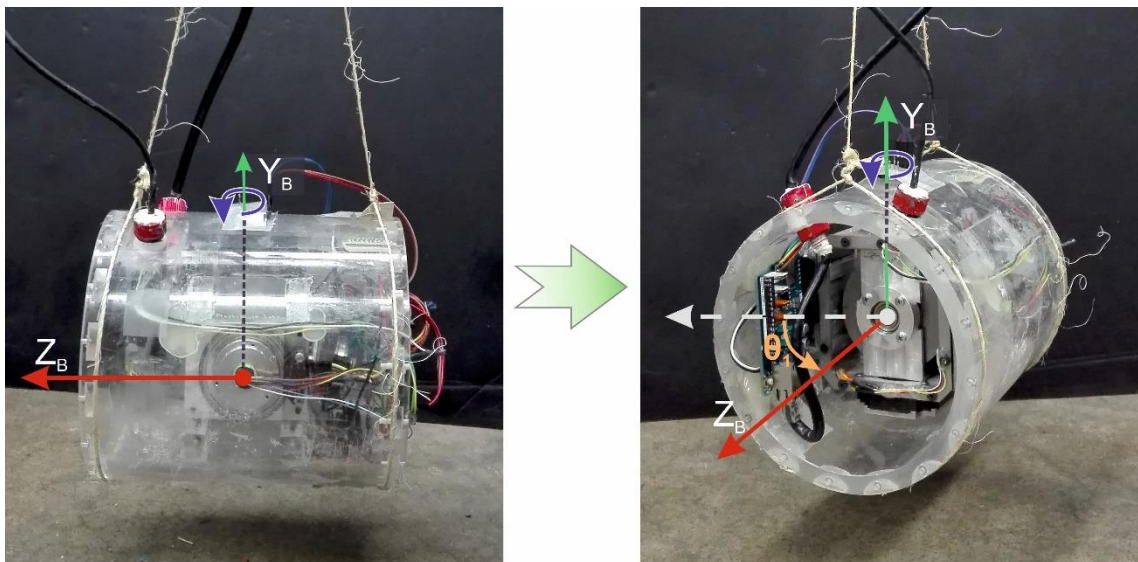


Figure 6-1: One DOF Hanging Method in Air

The body is hanged with two ropes connected to one single one on the top. The rope is fixed at the top of the room to minimize the influences of the hanging method on the dynamics of the body. The swimmer is hanged horizontal and the only DOF of the body

is the rotation around the vertical Y_B axis with the angle θ_1 ; whereby, the actual position of the swimmer is measured by the IMU.

6.1.2 Open-Loop Control

For an evaluation of the created forces by the center wheel, open-loop test have been done. Additional tests to show the agility of the two gimbals are performed. The tests are done with the hanging method shown before. Both, the reaction force and the gyro force of the center wheel shall be shown in this chapter.

6.1.2.1 Reaction Force

To measure the influence of the reaction force from the gyro to the body the outer gimbal is hold in vertical position and the inner gimbal rotates around the vertical Y_B axis. The center wheel rotates with 11,400 *RPM* in each case. As described in chapter 2, when the inner gimbal rotates in one direction the body shall rotate in the other direction. The following figure shows the results regarding the open-loop test.

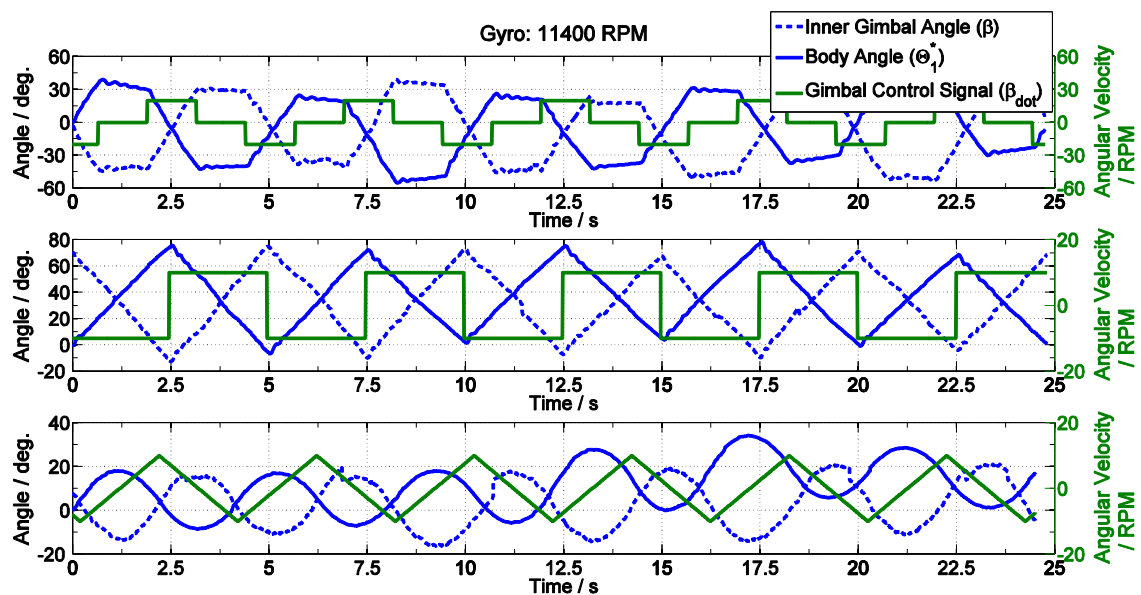


Figure 6-2: Results Hanged Open-Loop Test – Reaction Force

The first plot in the figure above shows a minus one zero plus one signal form, with an amplitude of 20 *RPM* and a frequency of 0.2 *Hz*. The second plot shows a rectangle input signal, with an amplitude of 10 *RPM* and a frequency of 0.25 *Hz*. The third plot shows a triangle input signal, with a maximal amplitude of 10 *RPM* and a frequency of 0.25 *Hz*. Each experiment is done with locked outer gimbal and rotating inner gimbal. It can be seen that the body movement θ_1 is because of the reaction torque of the high speed wheel in opposite direction of the gimbal rotation β . However, the input signal of the inner gimbal, with respect to the fuselage is bigger, then the effective body rotation, with respect to the ground.

6.1.2.2 Gyro Force

The second force, which is provided by the center wheel, is the gyro force. It acts perpendicular to the reactions force. It follows that for this case the inner gimbal is locked and the outer gimbal rotates. Expecting from physics, the gyro force is much bigger than the reaction force. Therefore, the amplitude of the input signal for the outer gimbal and the velocity of the gyro is reduced, Figure 6-3.

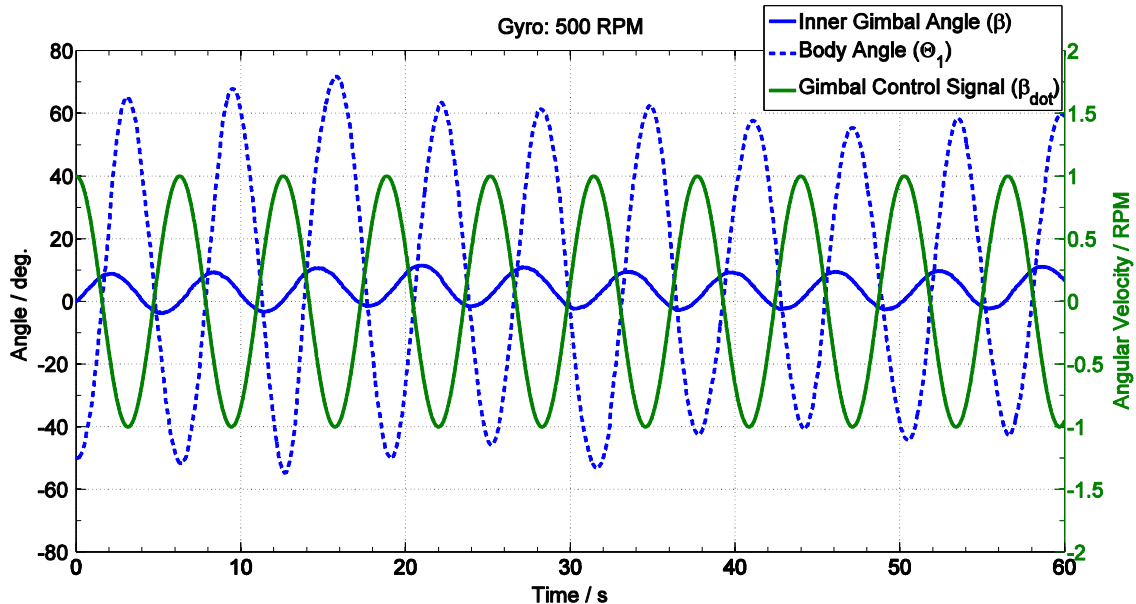


Figure 6-3: Results Hanged Open-Loop Test – Gyro Force

A sine wave with an amplitude of 1 *RPM* and a frequency of 0.15 *Hz* is chosen. As expected, even with a smaller input signal to the outer gimbal the movement of the body itself is much bigger. The body angle θ_1 moves for around 60 *deg* and the outer gimbal β moves just by 6 *deg*. It is important to note that the gyro speed in this case is just 500 *RPM*, which is the lowest possible velocity and about 4.5% of the speed used for the experiment in chapter 6.1.2.1.

6.1.3 Close-Loop Control – Yaw Angle

In a further step, a simple close-loop feedback control model with a proportional controller ($K_p = 0.5$) and with an input saturation (-10RPM to 10RPM) is introduced. The same hanging method as in the previous chapter is used. The aim of the feedback control loop is to control the hanged body to a desired body by the usage of the reaction force. Therefore, the feedback control loop consists of the desired angle θ_1^* , the measured actual body angle θ_1 and for the control variable the stepper input β . Additionally, the gimbal angle β is measured. Figure 6-4 shows the result for a simple repeating step signal. In the upper chart, the desired body angle θ_1^* and actual body angle θ_1 can be

6. Swimmer Agility Tests

seen. Further, in the lower chart the gimbal position β and the control input $\dot{\beta}$ for the stepper driver are shown.

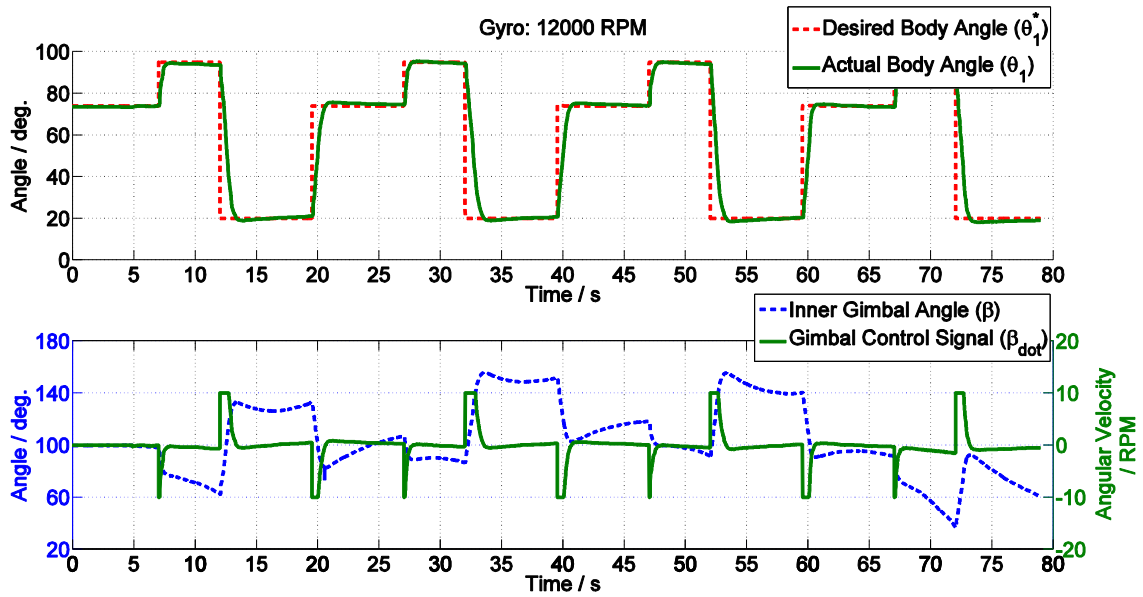


Figure 6-4: Results Hanged Close-Loop Test – Yaw Angle – Step Signal

It can be seen that the body follows the desired trajectory extremely fast without overshoot, which is a consequence of the I -behavior of the system and the used P-Controller. Regarding the torque applied by the handing method, the gimbal always corrects the body position while holding a certain body angle θ_1^* . The limits of the DGC MG can be pushed even further by superposing a step function by a sine wave (after 18 sec.), Figure 6-5.

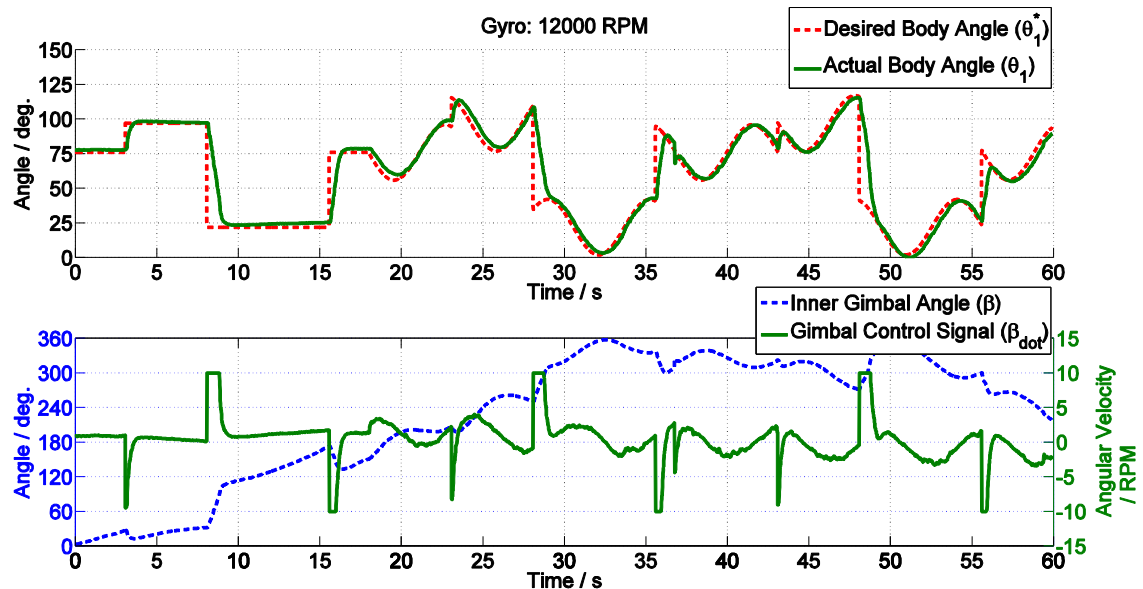


Figure 6-5: Results Hanged Close-Loop Test – Yaw Angle – Step Signal with Superposed Sine Wave

Even a complex signal such as the repeating step signal with the superposed sine wave can be followed accurate by the simple close-loop feedback controller. The velocity of the swimmer could be even pushed further by decreasing the saturation value.

Furthermore, note that by using the non-linear Backstepping controller introduced in chapter 3 the agility of the swimmer can be increased further.

6.2 Two Axis Agility Results

The next step, before submerging the swimmer, is to test the DGCMG with two DOF enabled. Therefore, a new hanging method has to be elaborated. To control the attitude of the body in air with two DOF the full dynamics non-linear controller is needed. For a first step, the performance of the outer gimbal has to be evaluated. Therefore, the new hanging method is used to perform a close loop test to control the pitch position of the swimmer. This chapter shows the new hanging method and the results of the one DOF test for the pitch angle.

6.2.1 Hanging Method for Two DOF

A new hanging method has to be elaborated to enable two DOF hanging tests. A metal ring is mounted around the body of the swimmer, whereby two fixings in direction of the X_B axis are mounted. At these fixings the wire is hooked and enables the rotation around the body X_B axis, for pitch rotations, and rotations around the Y_B axis to enable yaw rotations. The hanging method can be seen in the picture below.

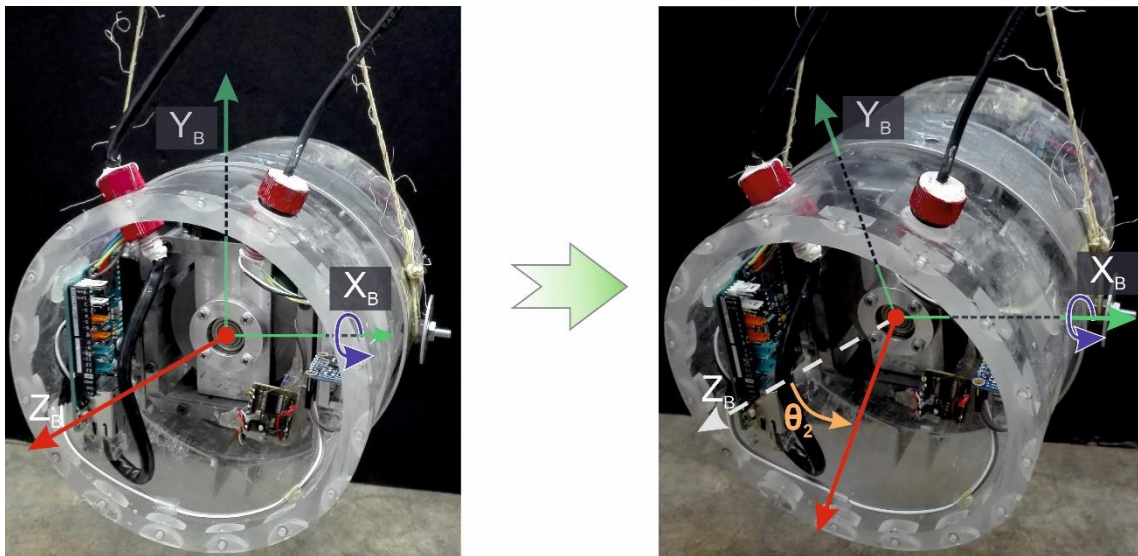


Figure 6-6: Two DOF Hanging Method in Air

It can be seen that the body rotates around the horizontal X_B axis with the angle θ_2 . The rotation around the vertical Y_B axis with the angle θ_1 can be seen in Figure 6-1. The new hanging method enables the possibility to test the DGCMG in two DOF.

6.2.2 Close-Loop Result – Pitch Angle

To control the pitch angle of the swimmer the inner gimbal is fixed in the $x - y$ plane and the outer gimbal rotates around the x -axis. This allows to control the swimmers pitch

angle and to evaluate the performance of the gimbal. Therefore, the pitch angle θ_2 is measured by the IMU and the desired body angle θ_2^* is fed into the simple P-feedback controller, which calculates the control signal for the outer gimbal velocity $\dot{\alpha}$. Figure 6-7, shows the results of the test run.

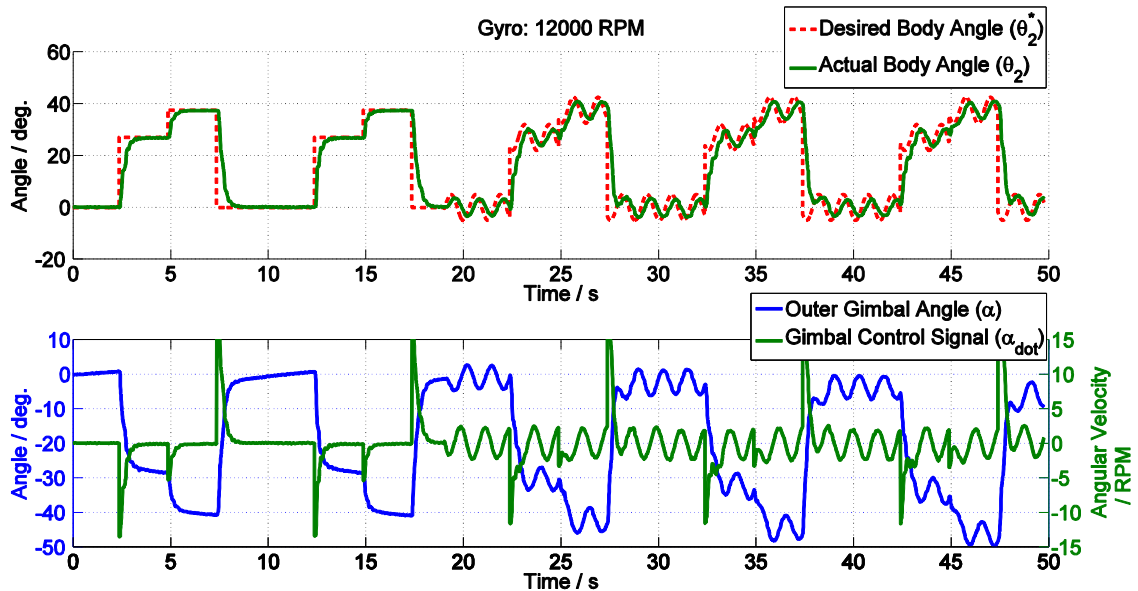


Figure 6-7: Results Hanged Close-Loop Test – Pitch Angle – Step Signal with Superposed Sine Wave
It can be seen that the torque generated by the DGCMG is high enough to rotate the swimmer and hold its position. Even by superposing the step signal with a small-amplitude sine wave the swimmer is following the desired position. In air, the maximal angle is limited because on a certain point the torque provided by the gyro is not strong enough to rotate the body.

6.3 Results Evaluation

It is shown that the DGCMG is strong enough to rotate the swimmer in air. The outer and the inner gimbal are working accurate and are able to rotate fast enough to control the yaw-position and the pitch-position. By using a simple feedback control-loop with a P-controller, the position can be controlled easily. This control method has one disadvantage. It is only possible to control one DOF simultaneously, because the reaction and gyro force vector is strongly dependent from both gimbal position. However, to control the multiple DOF of the swimmer the non-linear controller, introduced in chapter 3.2, has to be used.

7. Conclusions

7.1 Closure

The new design, elaboration of the dynamics, the control and the instrumentation of a super-maneuverable unmanned underwater vehicle is shown in this work. The orientation of the swimmer is controlled without taking benefit of thrusters, wings, rudders, stabilizers or fins. A Double Gimbal Control Moment Gyro (DGCMG) generates an inertia platform, which is used to rotate the swimmer in water. Therefore, two individually controllable gimbals are used to rotate the swimmer around the inertia platform. Two counter-rotating thrusters generate the thrust for the forward and backward propulsion. Further, the dynamic model has been introduced. This model is based on four connected ridged bodies. By the usage of Newton-Euler methods and a few assumptions made, the dynamics was derived. Finally, a non-linear controller based on a Lyapunov function with Backstepping approach has been introduced for the attitude regulation and reorientation control of the swimmer. Whereby, a combination of the reaction and gyro force generated by the control moment gyro and the torques produced by the thrusters are used. Next, the instrumentation of the swimmer is shown, which includes the control of the sensors and actuators by the embedded system and the implementation of the control algorithms. These control algorithms are processed in MATLAB Simulink, which just exchanges the control signals with the embedded system. Finally, different open-loop and close-loop test have been performed. The open-loop tests showed that depending on which gimbal is rotated either a reaction force or a gyro force acts on the body of the swimmer. Whereby, the gyro force is much bigger than the reaction force at the same gyro speed. Close-loop tests are performed to control the yaw-angle and the pitch-angle of the swimmer in one dimension and to show the performance of the individual gimbals of the double gimbal control moment gyro system. These tests proof that the DGCMG is powerful enough to perform these reorientation maneuvers and the high maneuverability of the system.

7.2 Future Work

Future tasks should deal with the performance testing of the robot in water, and the full dynamic non-linear controller for full 3D orientation maneuvers shall be implemented. The final aim is to use both the thrusters and the gyros system to reorientate the swimmer in water. After proofing the functionality of the developed system in water, the final control algorithms shall be implemented on the embedded system and a battery power supply shall enable full autonomy.

Bibliography

- [1] R. Alam, S. Mohsen, P. W. Grenfell, S. Messner und M. A. Jalali, „Supermaneuverable Autonomous Swimmer,“ in *31st Symposium on Naval Hydrodynamics*, Monterey, CA, USA, 2016.
- [2] „Berkeley - TAF Lab,“ [Online]. Available: <http://taflab.berkeley.edu/>. [Zugriff am 13. 06. 2016].
- [3] M. Saadat und M. A. Jalali, *Design and Prototyping of an Attitude Control*, Sharif University of Technology, 2009.
- [4] R. Votel und D. Sinclair, „A new active gyrostabiliser system for ride control of,“ in *26th Annual AIAA/USU Conference on Small Satellites*, Logan, UT, USA.
- [5] „Cubli,“ ETH Zurich, [Online]. Available: <http://www.idsc.ethz.ch/research-dandrea/research-projects/cubli.html>. [Zugriff am 14. 07. 201].
- [6] N. Townsend, A. Murphy und R. S. Shenoi, „A new active gyrostabiliser system for ride control of,“ *Ocean Engineering*, 2006.
- [7] J. G. Bellingham, „Platforms: Autonomous Underwater Vehicles,“ in *Encyclopedia of Ocean Sciences (Second Edition)*, Oxford, 2009.
- [8] M. Barisic, N. Miskovic und Z. Vukic, „A Measure of Quality of Control for 2D AUV Formations,“ *IFAC Proceedings Volumes*, Nr. 45, pp. 299-306, 2012.
- [9] J. Diebel, „Representing Attitude: Euler Angles, Unit Quaternions, and Rotation,“ Stanford University, CA, USA, 2006.
- [10] M. Krstic , I. Kanellakopoulos und P. V. Kokotovic, *Nonlinear and Adaptive Control Design*, Wiley-Interscience, 1995.
- [11] H. K. Khalil, *Nonlinear Systems*, Pearson, 2001.
- [12] A. Behal, D. Dawson, E. Zergeroglu und Y. Fang, „Nonlinear Tracking Control of an Underactuated,“ in *Proceedings of the American Control Conference*, Anchorage, AK, USA, 2002.
- [13] „Arduino Genuino,“ [Online]. Available: <https://www.arduino.cc/>. [Zugriff am 06. 04. 2016].
- [14] „Arduino Genuino Uno Board,“ [Online]. Available: <https://www.arduino.cc/en/Main/ArduinoBoardUno>. [Zugriff am 10. 07. 2016].
- [15] „Arduino Mega 2560 Board,“ [Online]. Available: <https://www.arduino.cc/en/Main/ArduinoBoardMega2560>. [Zugriff am 06. 04. 2016].

- [16] „Arduino Due Board,“ [Online]. Available:
<https://www.arduino.cc/en/Main/ArduinoBoardDue>. [Zugriff am 06. 04. 2016].
- [17] „Arduino - Open-source Software (IDE),“ [Online]. Available:
<https://www.arduino.cc/en/Main/Software>. [Zugriff am 10. 07. 2016].
- [18] „Sanyo Pancake Stepper Motor,“ [Online]. Available:
<https://www.pololu.com/product/2299>. [Zugriff am 18. 03. 2016].
- [19] „Adafruit Motor Shield V2 for Arduino,“ [Online]. Available:
<https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/overview>. [Zugriff am 16. 03. 2016].
- [20] „Pololu Stepper Motor Driver A4988,“ [Online]. Available:
<https://www.pololu.com/product/1182>. [Zugriff am 18. 03. 2016].
- [21] „Trinamic TMC2100 - SilentStepStick Stepper Motor Driver,“ [Online]. Available:
<http://www.watterott.com/index.php?page=product&info=4107>. [Zugriff am 26. 05. 2016].
- [22] „SparkFun MiniGen - Signal Generator Shield,“ [Online]. Available:
<https://learn.sparkfun.com/tutorials/minigen-hookup-guide>. [Zugriff am 18. 03. 2016].
- [23] „Arduino Library - DueTimer,“ [Online]. Available:
<https://github.com/ivanseidel/DueTimer>. [Zugriff am 18. 03. 2016].
- [24] „US Digital EM1 Transmissive Optical Encoder Module - EM1-2-1800-I,“ [Online]. Available:
<http://www.usdigital.com/products/encoders/incremental/modules/EM1>. [Zugriff am 06. 04. 2016].
- [25] „US Digital HUBDISK-2 2" Transmissive Rotary Disk,“ [Online]. Available:
<http://www.usdigital.com/products/encoders/incremental/rotary/disks/HUBDISK-2>. [Zugriff am 06. 04. 2016].
- [26] „Arduino Playground - Reading Rotary Encoders,“ [Online]. Available:
<http://playground.arduino.cc/Main/RotaryEncoders>.
- [27] „Robogaia 3 Axis Encoder Counter Shield,“ [Online]. Available:
<http://www.robogaia.com/3-axis-encoder-conter-arduino-shield.html>. [Zugriff am 08. 07. 2016].
- [28] „Anaheim Automation - Brushless DC Motor BLU09,“ [Online]. Available:
<http://www.anaheimautomation.com/products/brushless/brushless-motor-item.php?SID=296&pt=i&tID=96&cID=22>. [Zugriff am 10. 03. 2016].

- [29] „Anaheim Automation - Brushless Speed Controller MDC010-024031,“ [Online]. Available: <http://www.anaheimautomation.com/products/brushless/brushless-driver-controller-item.php?slD=350&serID=15&pt=i&tID=999&clD=23>. [Zugriff am 10. 03. 2016].
- [30] „BlueRobotics Thruster T100,“ [Online]. Available: <https://www.bluerobotics.com/store/thrusters/t100-thruster/>. [Zugriff am 10. 03. 2016].
- [31] „BlueRobotics Controller ESC 30A,“ [Online]. Available: <https://www.bluerobotics.com/store/electronics/besc-30-r1/>. [Zugriff am 10. 03. 2016].
- [32] „Arduino Library - Servo.h,“ [Online]. Available: <https://www.arduino.cc/en/Reference/Servo>. [Zugriff am 10. 07. 2016].
- [33] „Adafruit IMU - BNO055,“ [Online]. Available: <https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview>. [Zugriff am 12. 04. 2016].
- [34] „Adafruit 4-channel I2C-safe Bi-directional Logic Level Converter,“ [Online]. Available: <https://www.adafruit.com/product/757>. [Zugriff am 06. 04. 2016].
- [35] „Arduino Library - Wire,“ [Online]. Available: <https://www.arduino.cc/en/Reference/Wire>. [Zugriff am 08. 07. 2016].
- [36] „Arduino Playground - Interfacing MATLAB Simulink,“ [Online]. Available: <http://playground.arduino.cc/Interfacing/Matlab>. [Zugriff am 12. 04. 2016].
- [37] „Sparkfun Arduino Ethernet Shield 2,“ [Online]. Available: <https://www.sparkfun.com/products/11166>. [Zugriff am 12. 04. 2016].

List of Figures

Figure 1-1: Super-Maneuverable Unmanned Underwater Vehicle (SUUV).....	1
Figure 1-2: “Cubli” – ETH Zurich Project.....	3
Figure 1-3: Super-Agile Swimmer – Optical Communication Network with Submarine	4
Figure 2-1: Super-Maneuverable Unmanned Underwater Vehicle Design	5
Figure 2-2: Double Gimbal Control Moment Gyro (DGCMG).....	6
Figure 2-3: Double Gimbal Control Moment Gyro – Coordinate Systems	8
Figure 4-1: Arduino Genuino Uno.....	18
Figure 4-2: Arduino Mega 2560 Board.....	19
Figure 4-3: Arduino Due Board.....	20
Figure 4-4: Arduino Open–Source Software (IDE) Layout	21
Figure 4-5: Sanyo Pancake Stepper Motor.....	23
Figure 4-6: Adafruit Motor Shield 2.....	24
Figure 4-7: Polulu A4988 – Stepper Motor Driver	25
Figure 4-8: Sanyo Pancake Stepper Motor with Polulu A4988 Driver – Wiring Diagram	25
Figure 4-9: Trinamic TMC2100 – Stepper Motor Driver	26
Figure 4-10: Sanyo Pancake Stepper Motor with Trinamic TMC2100 Driver – Wiring Diagram.....	27
Figure 4-11: Trinamic TMC2100 – Stepper Motor Driver – Active Current Control	28
Figure 4-12: SparkFun MiniGen – Signal Generator Shield.....	29
Figure 4-13: US Digital EM1 Transmissive Optical Encoder	31
Figure 4-14: Rotatory Encoder Channel Pulse Diagram	32
Figure 4-15: Robogaia Encoder Counter Shield	34
Figure 4-16: Speed Controller (a) and Brushless DC Motor (b)	36
Figure 4-17: Brushless DC Motor with Speed Controller – Gyro – Wiring Diagram	36
Figure 4-18: BlueRobotics T100 Thruster (a) and ESC Motor Controller (b)	37
Figure 4-19: BlueRobotics T100 Thruster with ESC Motor Controller – Wiring Diagram.....	38
Figure 4-20: Adafruit BNO055 Absolute Orientation Sensor	39
Figure 4-21: Schematic – Arduino/Arduino Communication.....	41
Figure 4-22: Sparkfun Arduino Ethernet Shield 2	45
Figure 4-23: Schematic – Arduino MATLAB Communication.....	45
Figure 4-24: Arduino Ethernet Connection – MATLAB Function in Simulink.....	51
Figure 4-25: Flow Chart – Software Task Distribution.....	52
Figure 4-26: Electrical Circuit and Logic Connections of the SUUV Electronics....	53
Figure 4-27: Power Supply of the SUUV Electronics	54
Figure 4-28: Logic Signal Connections of the SUUV Ectronics.....	55
Figure 5-1: Front View of the Assembled Swimmer	57
Figure 5-2: Rear View of the Assembled Swimmer.....	58
Figure 5-3: Side View of the Assembled Swimmer	59
Figure 6-1: One DOF Hanging Method in Air.....	60
Figure 6-2: Results Hanged Open-Loop Test – Reaction Force	61

Figure 6-3: Results Hanged Open-Loop Test – Gyro Force.....	62
Figure 6-4: Results Hanged Close-Loop Test – Yaw Angle – Step Signal	63
Figure 6-5: Results Hanged Close-Loop Test – Yaw Angle – Step Signal with Superposed Sine Wave	63
Figure 6-6: Two DOF Hanging Method in Air.....	64
Figure 6-7: Results Hanged Close-Loop Test – Pitch Angle – Step Signal with Superposed Sine Wave	65
Figure A-1: Motor Driver Circuit – Polulu A4988	XV
Figure A-2: Motor Driver Circuit – Trinamic TMC2100	XVI
Figure A-3: Motor Driver Circuit – BlueRobotics ESC	XVII
Figure A-4: Motor Driver Circuit – BLDC Motor Speed Controller	XVIII
Figure B-1: Electric Circuit – Signal Connections	XIX
Figure B-2: Electric Circuit – Logic Signal Connections	XX
Figure B-3: Electric Circuit – Power Supply	XXI
Figure C-1: Software Tasks – Flowchart.....	XXII

List of Listings

Listing 4-1: Arduino Object-Oriented Programming	22
Listing 4-2: Arduino Function – Direct Digital Write.....	29
Listing 4-3: Stepper Driver – Attach stepperClock functions	30
Listing 4-4: Stepper Driver – Direct Pulse Generation	30
Listing 4-5: Arduino Encoder Direct – Attach Interrupt.....	32
Listing 4-6: Arduino Encoder Direct – Position Read	33
Listing 4-7: Arduino Encoder Direct – Position Reset	33
Listing 4-8: Arduino Function – Direct Digital Read.....	33
Listing 4-9: Arduino Encoder Shield – Initialization	34
Listing 4-10: Arduino Encoder Shield – Read Encoder Value.....	35
Listing 4-11: Arduino Encoder Shield – Reset Counter.....	35
Listing 4-12: Arduino Library – Servo.h Modification.....	38
Listing 4-13: Arduino DueTimer Library – Servo Library Compatibility	38
Listing 4-14: Adafruit IMU – Include Libraries	39
Listing 4-15: Adafruit IMU – Setup Procedure.....	40
Listing 4-16: Adafruit IMU – Read Absolute and Raw Orientation.....	40
Listing 4-17: Master Arduino – Wire1 Initialization	42
Listing 4-18: Master Arduino – Request Data from Slave	43
Listing 4-19: Slave Arduino – Wire Initialization	43
Listing 4-20: Slave Arduino – Buffer Generation.....	43
Listing 4-21: Slave Arduino – onRequest Function	44
Listing 4-22: MATLAB Ethernet Communication – Command String	46
Listing 4-23: Arduino Ethernet Communication – Command String	46
Listing 4-24: MATLAB Ethernet Communication – Block Generation.....	46
Listing 4-25: MATLAB Ethernet Communication – Setup	47
Listing 4-26: MATLAB Ethernet Communication – Initialization	47
Listing 4-27: MATLAB Ethernet Communication – Update	48
Listing 4-28: MATLAB Ethernet Communication – Terminate.....	48
Listing 4-29: Arduino Ethernet Connection – Ethernet Objects	49
Listing 4-30: Arduino Ethernet Connection – Ethernet Variables	49
Listing 4-31: Arduino Ethernet Connection – Setup Ethernet Communication	49
Listing 4-32: Arduino Ethernet Connection – Update Ethernet.....	50
Listing 4-33: Arduino Ethernet Connection – Split Command String	51

List of Abbreviations

Cal	University of California, Berkeley
SUUV	Super-Maneuverable Unmanned Underwater Vehicle
TAF Lab	Theoretical & Applied Fluid Dynamics Laboratory
ROV	Remotely Operated Vehicle
AUV	Autonomous Undersea Vehicle
UUV	Unmanned Underwater Vehicle
DGCMG	Double Gimbal Control Moment Gyro
CLF	Control Lyapunov Function
OG	Outer Gimbal
IG	Inner Gimbal
CPU	Center Processing Unit
BLDC	Brushless Direct Current
RMS	Root Mean Square
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
DOF	Degree of Freedom
I/O	Input/output
1D	One Dimensional
3D	Three Dimensional

List of Symbols

Symbol	Name	Unit
${}^a I_b$	Moment of Inertia of body b in coordinate system a	$kg \cdot m^2$
${}^a \boldsymbol{\omega}_b$	Angular Velocity of body b in coordinate system a	$rad \cdot s^{-1}$
${}^a \mathbf{H}_b$	Angular Momentum of body b in the coordinate system a	$kg \cdot m^2/s$
\mathbf{T}	Torque	$N \cdot m$
${}^b_a \mathbf{R}$	Euler Rotation Matrix from coordinate system a to b	–
J_w	Moment of Inertia of the center wheel with respect to the inner gimbal coordinate system	$kg \cdot m^2$
$\dot{\alpha}$	Angular Velocity of the outer gimbal in the outer gimbal coordinate system	$rad \cdot s^{-1}$
$\dot{\beta}$	Angular Velocity of the inner gimbal in the inner gimbal coordinate system	$rad \cdot s^{-1}$
Ω	Angular Velocity of the center wheel with respect to the inner gimbal coordinate system	$rad \cdot s^{-1}$
M	Torque generated by the thrusters	$N \cdot m$

A. Motor Driver Circuits

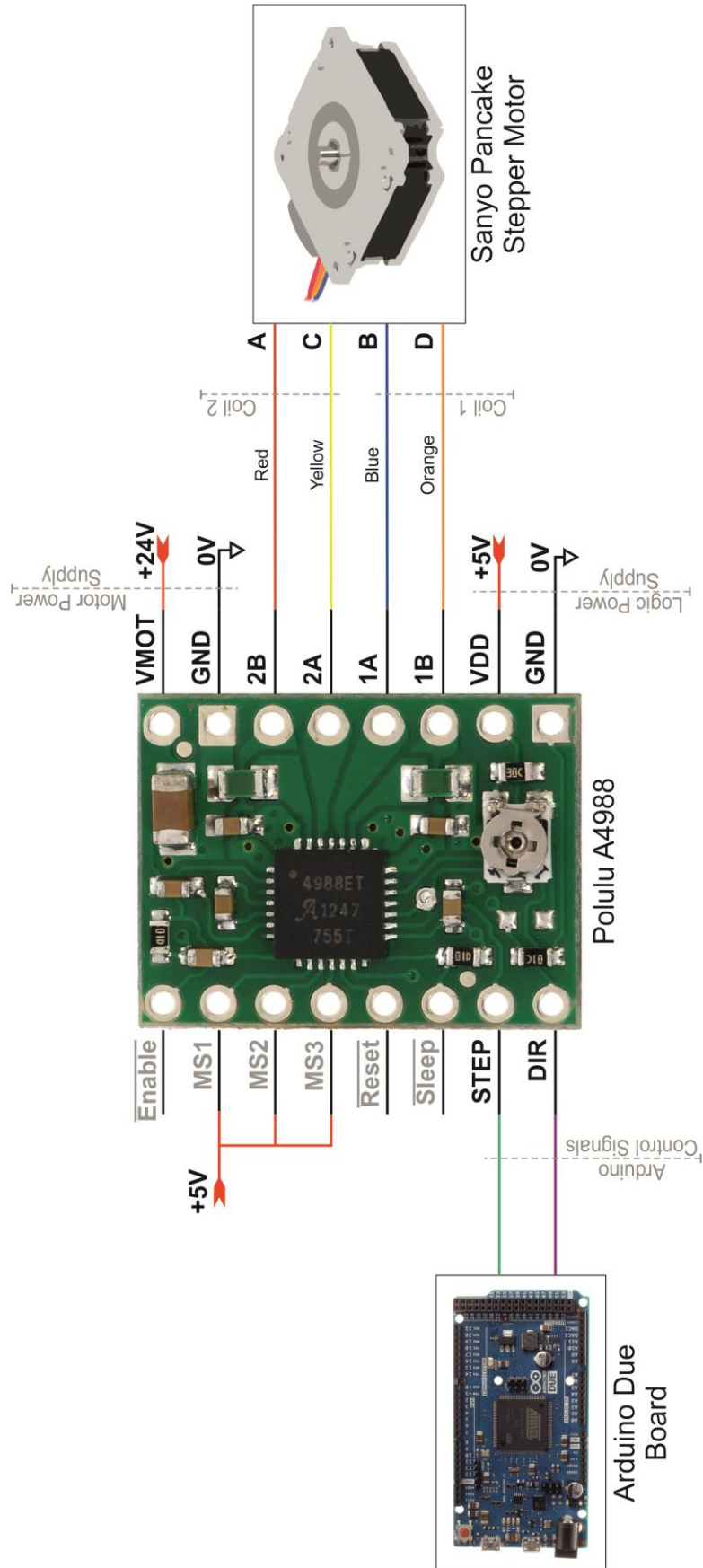


Figure A-1: Motor Driver Circuit – Polulu A4988

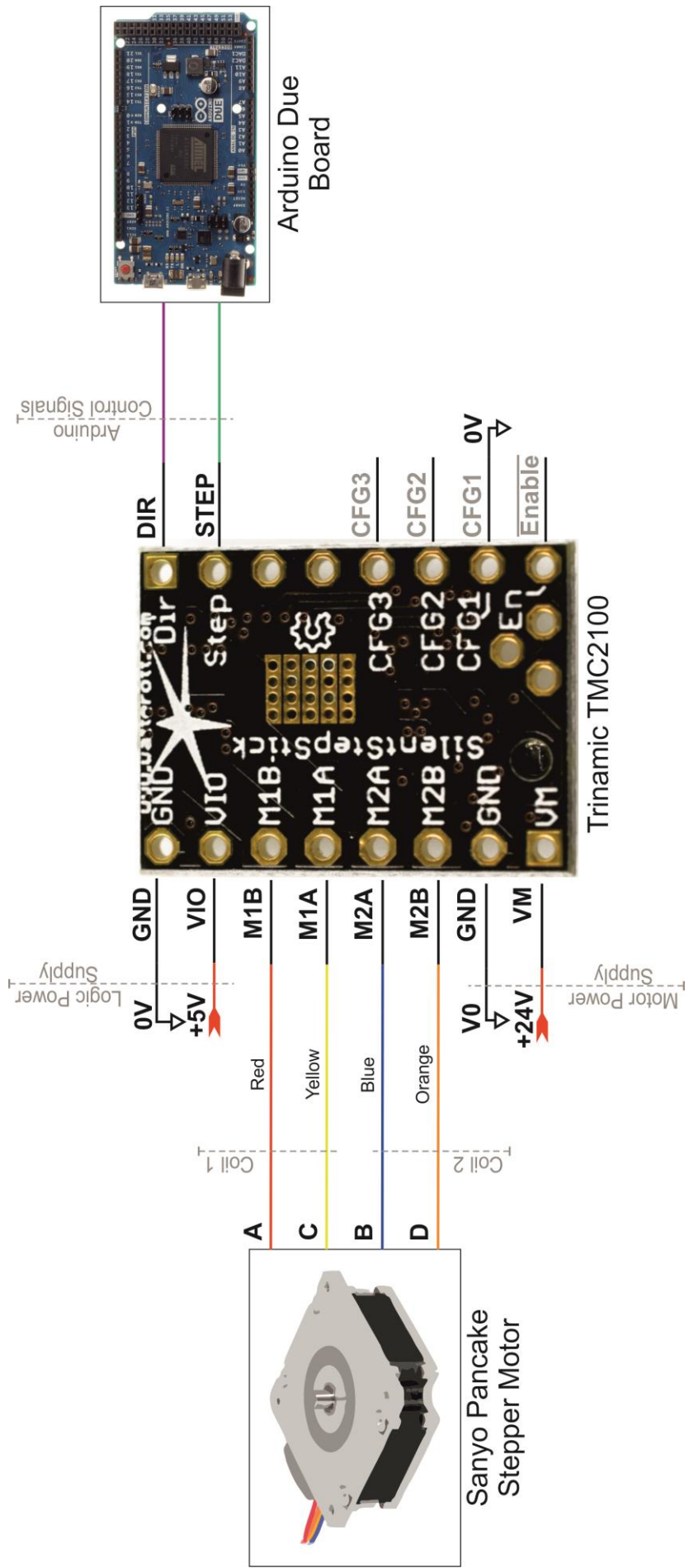


Figure A-2: Motor Driver Circuit – Trinamic TMC2100

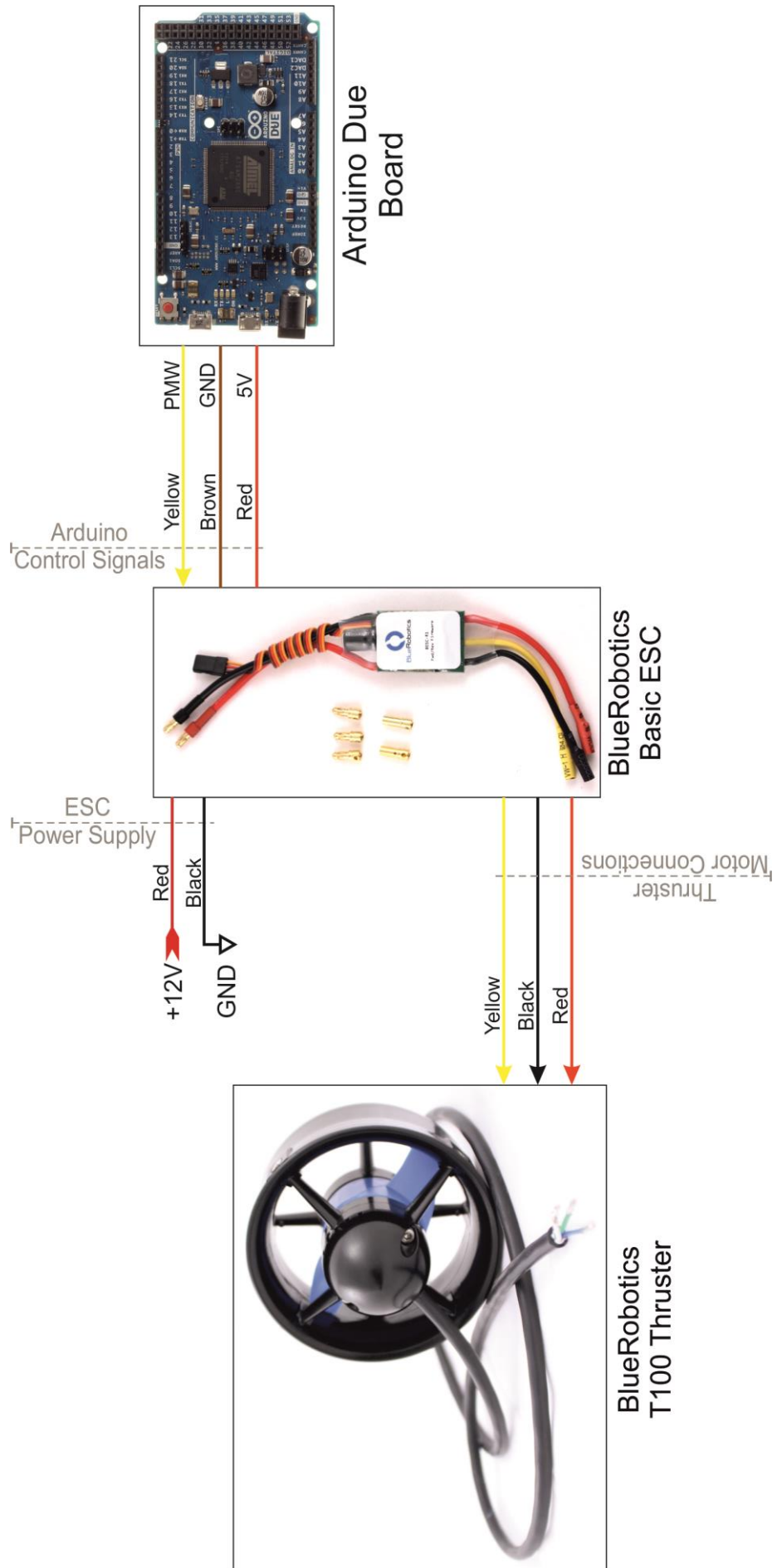


Figure A-3: Motor Driver Circuit – BlueRobotics ESC

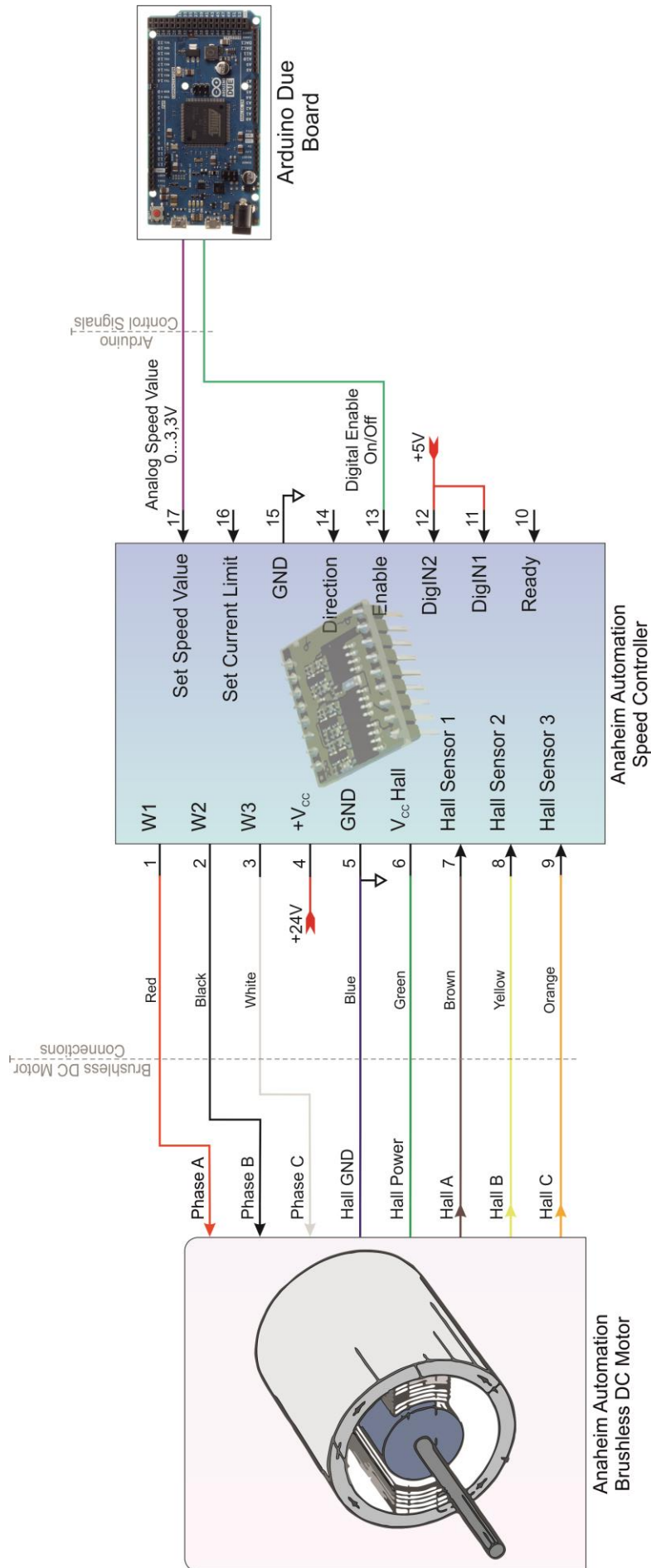


Figure A-4: Motor Driver Circuit – BLDC Motor Speed Controller

B. Electric Circuits

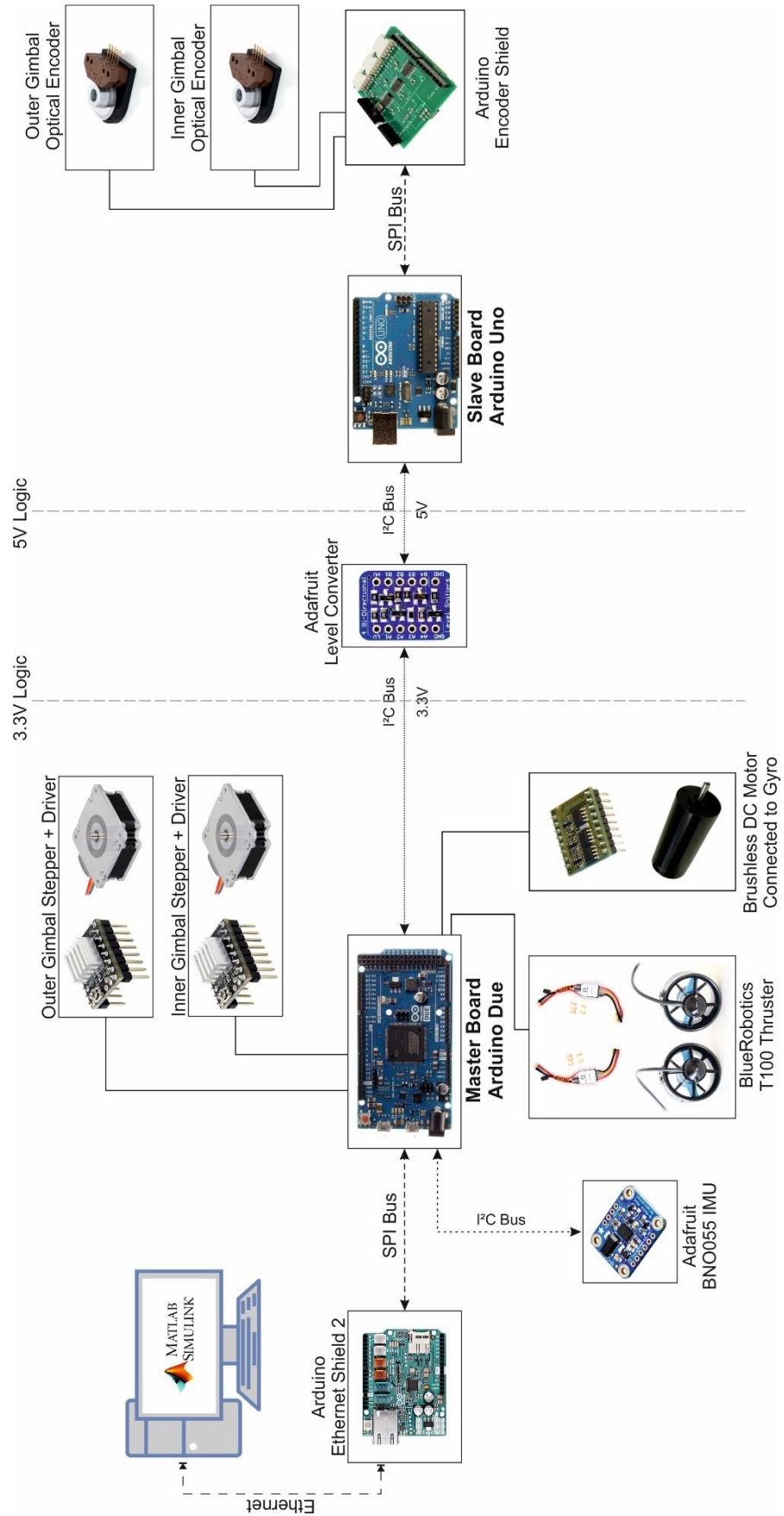


Figure B-1: Electric Circuit – Signal Connections

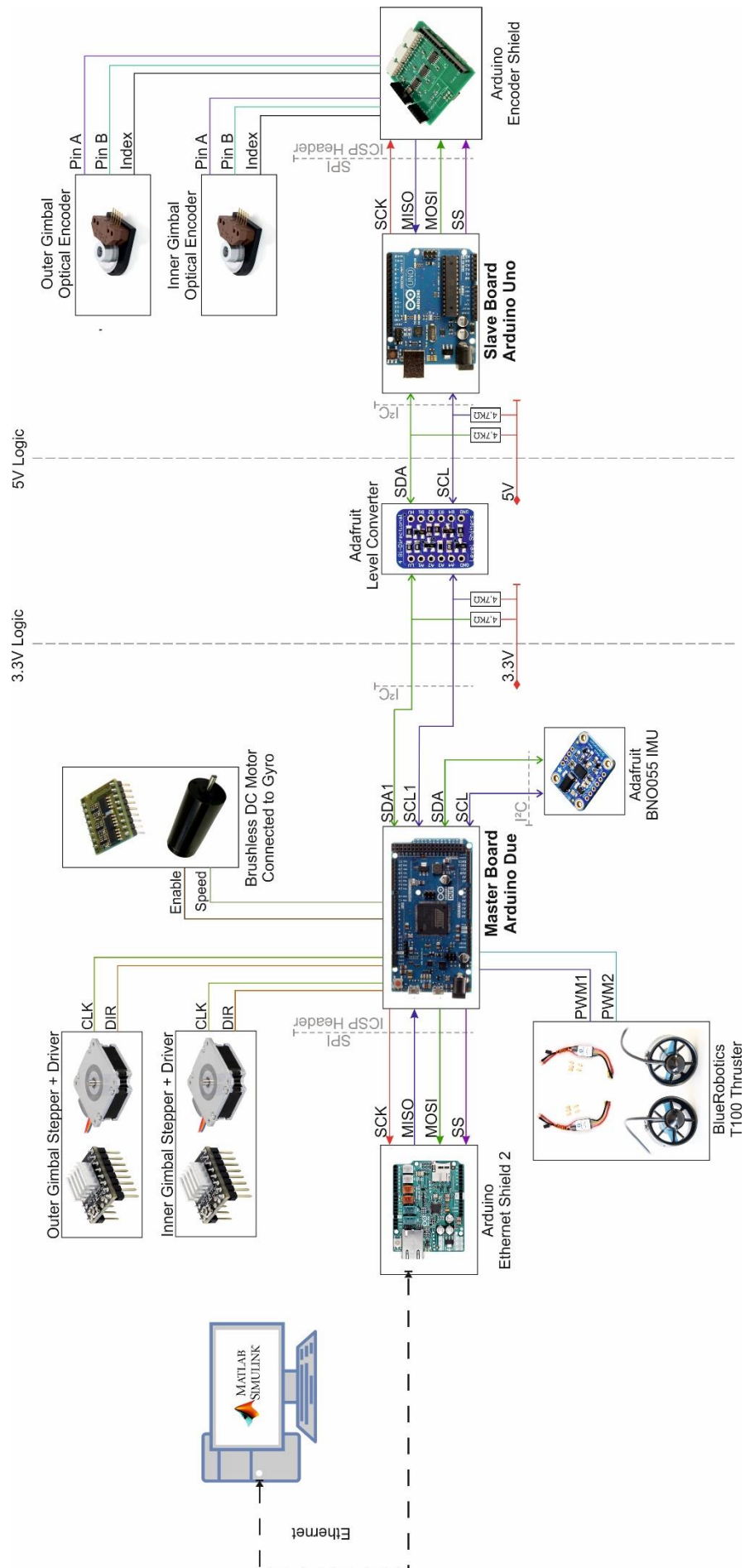


Figure B-2: Electric Circuit – Logic Signal Connections

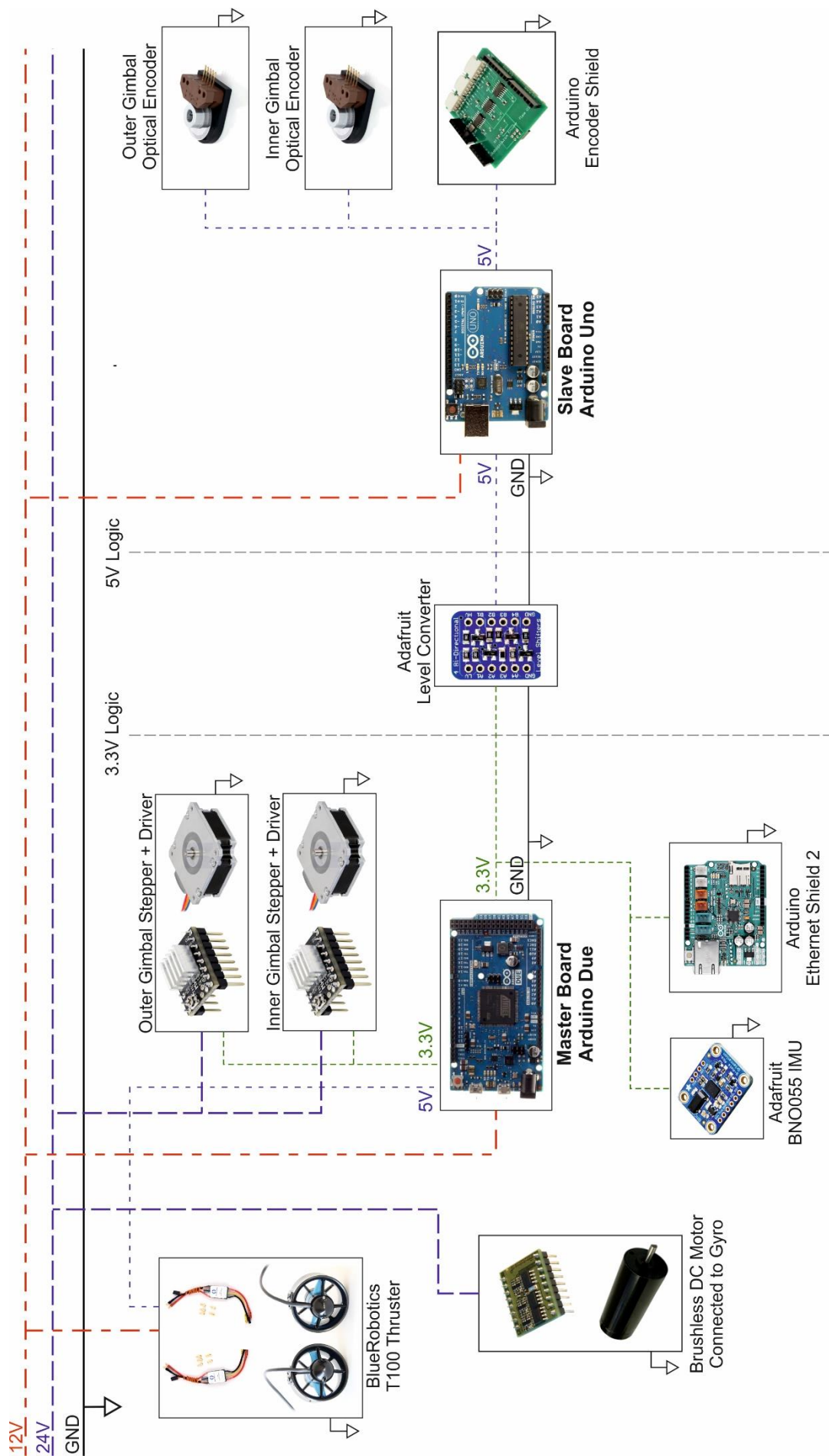


Figure B-3: Electric Circuit – Power Supply

C. Software Tasks

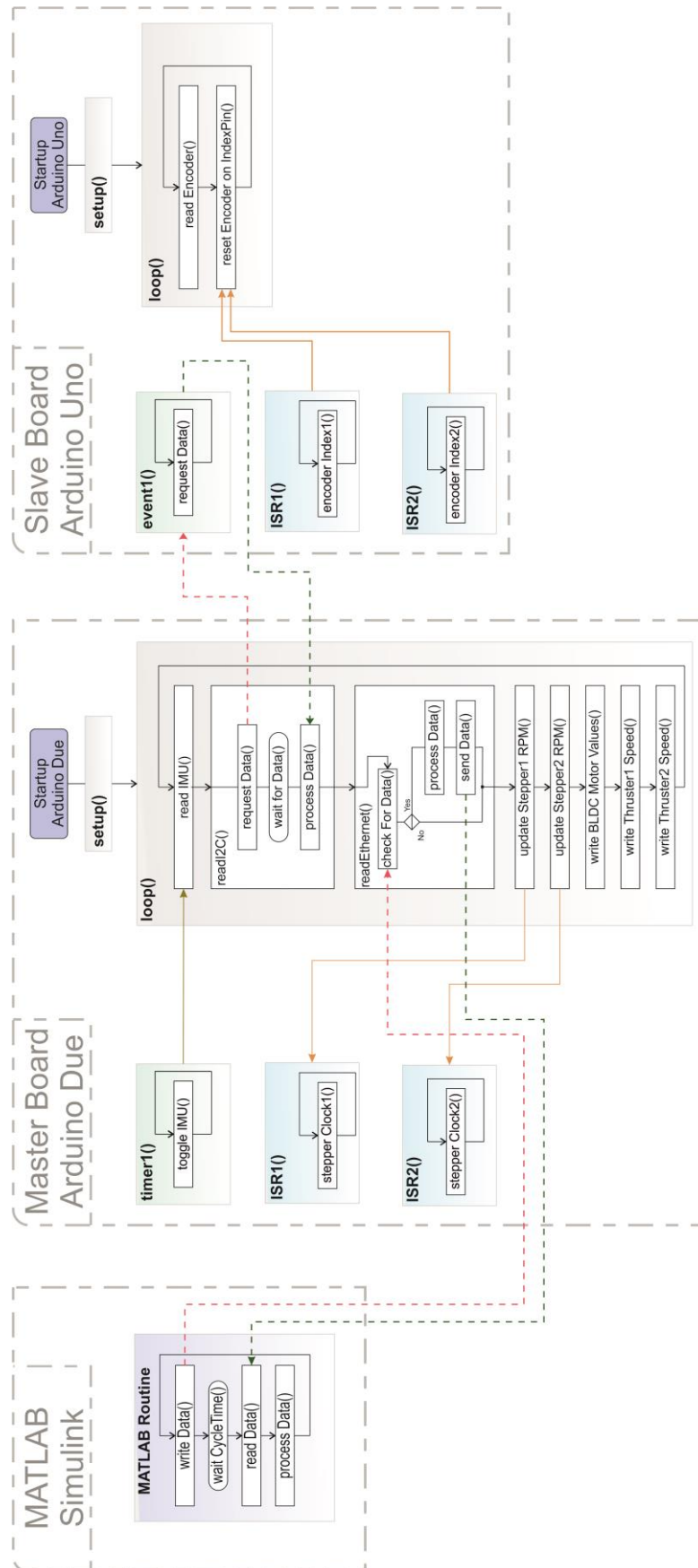


Figure C-1: Software Tasks – Flowchart