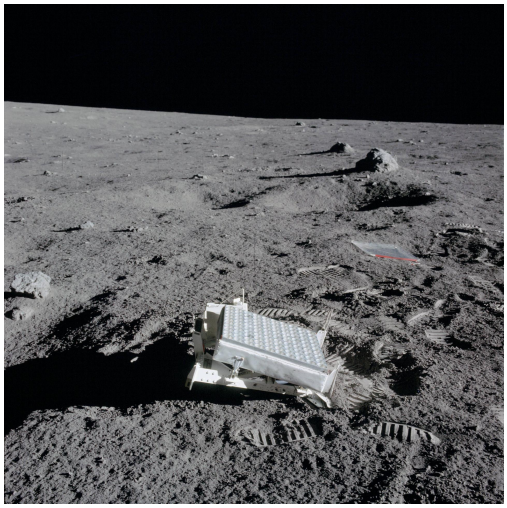
Research project Cubesat



ENES ERYIGIT
ALI UL HAQ
Amsterdam University of Applied Sciences

RECYCLE VALLEY

Supervisor: Nuwar Misconi

5 February 2017

Contents

Preface

Team project CubeSat would like to thank their project supervisor, Nuwar Misconi, for her consultancy and supervising. In addition, the group wants to thank Patrick de kok for additional advice and guidance during our project.

Introduction

In this report we will describe the creation of the CubeSat for Recycle Valley. Recycle valley is interested in retrieving the plastic bag from the moon using a CubeSat. (1). Before doing this a proof of concept of the final phase, the landing has to be defined and developed. This has to fulfill certain requirements and specification. (2). After the specifications and the requirements have been defined a research is conducted for each CubeSat aspect (3) and this will result in the Functional Design. The Functional Design describes how each component and will relate to eachother and how it is implemented (4). One of the final phases is to test and simulate the CubeSat (5). A conclusion and recommendation can be made based upon the findings of the project. (6 & 7)

Chapter 1

Background

Recycle valley is interested in retrieving the plastic bag in which the mirror for the Lunar Laser Ranging experiment was taken out of. They want to send a rocket to the moon containing a CubeSat that is able to land on the moon and retrieve the bag (1.1). The goal is to develop a proof of concept of a CubeSat free-falling into the water and stabilise itself (1.2). For this project research questions are used to aid in forming the scope and answering the main question (1.3.1)

1.1 Problem definition

Recycle Valley, located in Amsterdam, is interested in retrieving a plastic dust cover from the moon with a CubeSat. Every space mission is divided into several phases. One of the most critical phases is the landing of the spaceship. To provide a proof of concept Recycle Valley wants to simulate this landing on the moon by making a CubeSat have a free-fall dive into the water and stabilise itself for landing.
The following main question needs to be answered for this project: ”Is it possible to simulation and reproduce the landing on the moon using a CubeSat in water?”.

1.2 Goal

The goal of this project is to design a CubeSat that is able to survive a free-fall dive into the water and is able to stabilise itself and land in water.

1.3 Research question

Research questions aid in forming the scope of the project and answering the main question. The research question can be divided into two topics:

1. Construction
2. Electronics

1.3.1 Construction

- What are design requirements of a CubeSat?
- What is applicable for the project?
- Have there been projects similar to this projects?
    - What was the purpose of their project?
    - Were they used in similar environments?
    - How did the construction and the design look and what materials did they use?
- Which material and construction are strong enough to withstand and survive a free-fall into the water?
- Are these material easily available?
- What type of techniques could be used to create the CubeSat?
- how accessible does the CubeSat need to be?
- How could the electronics be protected from water?
- What is the optimal location for the propulsion system?

- What is the mass of the CubeSat?
- What type of construction could be used for the landing gear?
- What is the maximum height at which the CubeSat will survive the fall?
- How long does the CubeSat need to stabilise in water?
    - What is the minimum water depth that is needed?
- Is there need for material protection?

1.3.2 Electronics

- What does the CubeSat need to measure of its environment and itself?
    - What does the CubeSat need to know in order to stabilise itself?
    - How will the CubeSat differentiate from being in air and in-water?
    - Which sensors are needed for the measurements?
    - Are the sensors readily available?
- How will the CubeSat measure the distance between the ground and itself?
- What kind of propulsion systems can be used?
- What kind of actuators does the CubeSat need to regulate the propulsion?
- How much energy does the CubeSat need to operate?
    - What kind of power source applicable?
    - What is the estimated energy consumption of the CubeSat?
- How will the CubeSat adjust and move under water?

Chapter 2

Specifications & requirements

In addition to the research questions in 1.3.1 certain specficiations & requirements will be defined, these will be divided in technical (2.1) and non-technical (2.2)

2.1 Technical requirements

- The entire CubeSat dimensions is 10x10x10 cm, according to the standard CubeSat specification (1U format) [12].
- The material and construction must survive the free-fall in water.
- The CubeSat can survive a free-fall dive into the water. Further specifications will follow after research.
- The CubeSat can stabilise underwater and land on its landing gear.
    - The CubeSat has enough processing power to calculate read sensor data and steer in real time.
    - The CubeSat has enough propulsion to stabilise and land.

2.2 Non-Technical requirements

- The budget for the CubeSat project is around €50 provided by Amsterdam University of Applied Sciences.

Chapter 3

Research

Based upon the research questions defined in 1.3.1 and the specifications and requirements the research can be conducted to find the feasibility of the specifications and requirements and create the proof of concept design of the CubeSat. This research is defined for the three aspects which are critical in creating the CubeSat. Before these three aspects will be described we will look at a general description of a CubeSat and if there have been CubeSat projects before similair to ours (3.1). After that the propulsion underwater (3.2), the construction of the CubeSat (3.3) and the electronics (3.4) will be described.

3.1 CubeSat research

CubeSats are a class of research spacecraft called nanosatellites. Nanosatellites are cheaper and smaller satellites, which can be used to enable mission CubeSats are built to standard dimensions of 10x10x10, this standard is denoted U. A 2U CubeSat will be twice as big, likewise for 3U or 6U CubeSats. CubeSats typically weigh less than 1.33 kg (3 lbs) per U. The CubeSat standards is summarised and shown in Appendix I [9]. From the CubeSat, the main requirement chosen is that the CubeSat structure has to be 10x10x10 cm. All other requirements are deemed to be too high extreme due to the fact that these standards are made for CubeSats launching into space, the CubeSat for the proof of concept will not be launched in space, these standards are therefore too strict and too hard to achieve in the limited span of time.

3.2 Propulsion

The CubeSat will free-fall in water, which means the propulsion mechanism has to work underwater. The main difference between water and air is the density in which the medium appears naturally, which is 1000 $\frac{kg}{m^3}$ for water at $4°$ Celcius and does not get denser as temperature rises. The air appears at 1,293 at $0°$ and does get denser as temperature rises but will never reach as close as the density of water [11][10]. In order to find propulsion mechanisms that may be feasible, we will look at propulsion mechanism used currently for CubeSats (found in [8]) and other mechanisms that were found and may be feasible.

3.2.1 Cold gas

A CubeSat can use a micro thruster to accelerate itself using cold gas as propulsion. It is the simplest configuration for a CubeSat because it requires the least components and is easier to design. The cold gas is released from a pressurised tank by a control valve and expelled through a thruster, which is the nozzle. By releasing the gas in a controlled manner it is possible to achieve different accelerations and allow the CubeSat to orientate itself. A schematic drawing of a cold gas system is shown below:

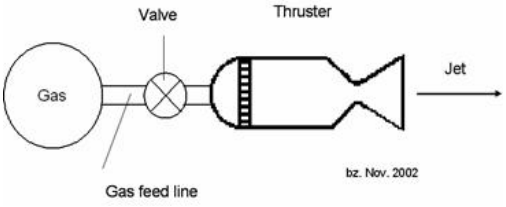


Figure 3.1: Cold gas thrust mechanism [13]

The equations for the force and torque provided by a thruster are:

$$\mathbf{F_{mexp}} = -\dot{m} * \mathbf{v_{rel}} \tag{3.1}$$

$$\mathbf{L_{mexp}} = \mathbf{r} * \mathbf{F_{mexp}} \tag{3.2}$$

Where $\mathbf{F_{mexp}}$ is the Force [N] of the expelled mass flow [$\frac{kg}{s}$] and $\mathbf{v_{rel}}$ is the velocity [$\frac{m}{s}$] of expelled mass relative to the spacecraft. Both of them are bold faces and are vectors denoting a magnitude and direction. The mass flux ($\dot{m}$) is a scalar, which is the rate at which mass is expelled.

Different types of cold gases have different thrust values and deliver a perform better or worse depending on its condition and needs. In the table below typical performance values for some cold gas propellants are shown (Table:3.1). A higher specific impulse would mean that the gas can achieve a higher force.

Propellant	Molecular Weight	Density	Specific Impulse [s]	
			Theoretical	Measured
Hydrogen	2.0	0.02	296	272
Helium	4.0	0.04	179	165
Nitrogen	28.0	0.28	80	73
Ammonia	17.0	Liquid	105	96
Carbon dioxide	44.0	Liquid	67	61

Table 3.1: Cold Gas Propellant Performances [1]

The use of Hydrogen is not recommended due to the fact that it is highly flammable, the other cold gases could be used to propel the CubeSat. This method is feasible for use in space. During the research, there have been no findings where this was used as an underwater propulsion mechanism. For propulsion under it would be too complex to design.

3.2.2 Reaction wheel

Reaction wheels is a method that is already been used in space and CubeSats. Three solid wheels in the CubeSat are powered by electric motors. The motor will spin the wheel into a certain direction and generate torque. The torque created by the spinning wheel will rotate the CubeSat in reverse direction for the same angular velocity (Figure 3.2).The combined angular velocity of both objects will be constant. This method is mostly combined with another mechanism due to the fact that the mass of the wheel limits the amount of torque that can be provided, therefore reaction wheels alone are not enough [8]

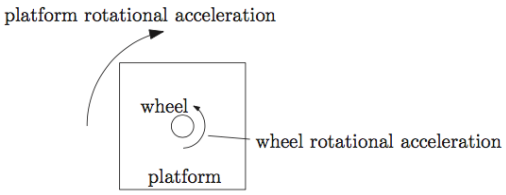


Figure 3.2: Reacion Wheel mechanism [13] [8]

$$I_1\omega_1 + I_2\omega_2 = const \tag{3.3}$$

3.2.3 Magneto-torque

The cube will rotate and stabilise by powering electromagnets in the CubeSat. This will attract or repel the CubeSat to the earth magnetic field and will allow the cube to regulate and stabilise its attitude. Like the reaction wheel, this mechanism alone is not sufficient for stabilisation and is mainly used for attitude corrections with small angles [8].

3.2.4 Balloon

If the balloon method is possible it will be the easiest method. This method uses a balloon filled with gas to make the CubeSat face the right direction and slow down its fall.
At the top of the CubeSat, there is a hole. The hole is there so the balloon can inflate and is connected to a gas tank with a valve. The microcontroller will send the signal to open the valve when we reach a determined height and a signal to close the valve when the balloon is inflated. The balloon will also slow down the fall and provide a safe landing. This method is applicable to air and water, but not in a vacuum because it needs resistance to slow down and change orientation.

3.2.5 Propellors

Because the concept will be thrown into the water, which is a different medium and more viscous than air it would be wise to choose propellors for the CubeSat stabilisation. All the above propulsion methods are too complex and difficult to create and execute underwater and mostly work in a vacuum to be effective, but by using propellers it would be possible to use the same configuration as the Cold gas method and still have a working method for underwater use. The underwater thruster or propellors would need to be free from the CubeSat structure to allow the water to flow around it. The propellor would then be connected to a motor from inside the CubeSat. This brings difficulty in the structure due to the fact that the connection between the shaft of the motor and the propeller needs to be rigid and strong enough to withstand a free-fall.

3.2.6 Water pump

The last propulsion mechanism is to used a water pump to drive the CubeSat underwater. A water pump will allow the CubeSat to be enclosed from its environment, with only the shafts of the motor going outwards. An example of the water pump can be seen in Figure 3.3. This design was used as inspiration for out concept. As seen in the figure the water pump is divided into three components. The impeller (propeller blades), the base structure and the top structure. The impeller is inserted into the base and driven by the motor and will allow to suck water through the top out to the exit nozzle[1]. A water pump mechanism is a feasible option primarily due to the fact that there will not be a lot of opening to allow water in the CubeSat. Also with different pump designs, different performance can be acquired. For our concept we can go with a simple design to proof that the water pump mechanism will work[2].

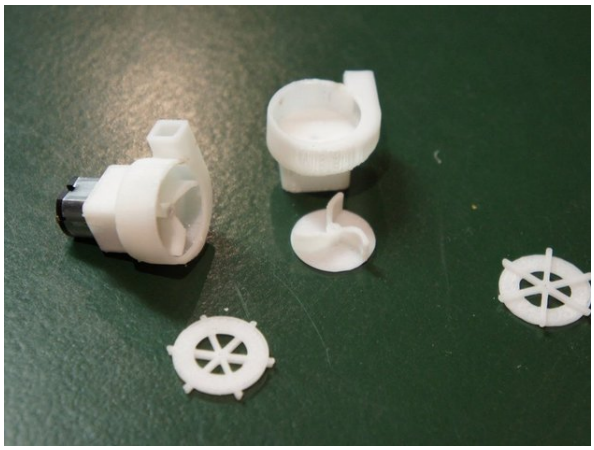


Figure 3.3: 3D printed water pump

[1] https://www.youtube.com/watch?v=tNMcnYSnQBU
[2] http://www.thingiverse.com/thing:4839

3.2.7 Conclusion

Based upon a research a possible method can be chosen for the propulsion underwater. Grades are given for each method with a range from 1 to 3. With 1 being the worst, 2 is average and 3 is great. Feasibility will denote the ability of us implementing this mechanism and the complexity that it has. Furthermore, the suitability will be looked upon for space and for water and looks at how this mechanism will perform in both of the environments. The method with the highest grade will be chosen as a propulsion mechanism.

Method	Feasibility	Suitable for Space	Suitable for Water	Total
Cold Gas	2	3	1	6
Reaction wheel	1	2	1	4
Magneto-torque	1	1	1	3
Balloon	1	1	2	4
Propellor	2	1	3	6
Water pump	3	1	3	7

Table 3.2: Components of CubeSat

From this table, it can be concluded that the water pump propulsion mechanism is the best choice for our concept.

3.3 Construction

The CubeSat will fall from a certain height and when the CubeSat reaches the water force will be exerted on the structure. The CubeSats created for space applications are used with Aircraft grade aluminium (Al 6061). Furthermore, they have an open construction whereas the CubeSat for this concept needs a watertight construction. Creating a cube with aluminium would mean that a metal sheet has to be folded to a cube and the joints of the sheets have to be connected using welding, another option for a metal construction is to use a solid cube and make the construction out of that. The first option does not give a guarantee of the joints being watertight and depends on the expertise of the maker, the second option is very expensive due to material costs of a solid cube.
A third option is with 3D printing, which allows the use of plastics such as PLA, Nylon and ABS. Based upon the advice of the innovation lab at the Hogeschool van Amsterdam it was advised to use PLA. With this advice, 3D printing is chosen as production process to create the CubeSat and its components.

With all of these methods, it is important to that the CubeSat must sink in water. It has to overcome the buoyant force (shown in Figure 3.4) according to the law of Archimedes [2]. "Archimedes Principle states that the buoyant force on a submerged object is equal to the weight of the fluid that is displaced by the object". The density of water is 1000 $\frac{kg}{m^3}$ or 1 $\frac{g}{cm^3}$, an object that weighs 1 gram will displace 1 cm^3. If the object weighs 25 grams it will displace 25 cm^3 of water, but if the volume of the object is 50 cm^3 it will only sink halfway. Therefore for the object to sink its density has to be higher than that of water. For a 10x10x10 cm CubeSat the density will be 1000 cm^3, The Cubesat has to weigh at least higher than 1000 gram or 1 kg for it to sink.

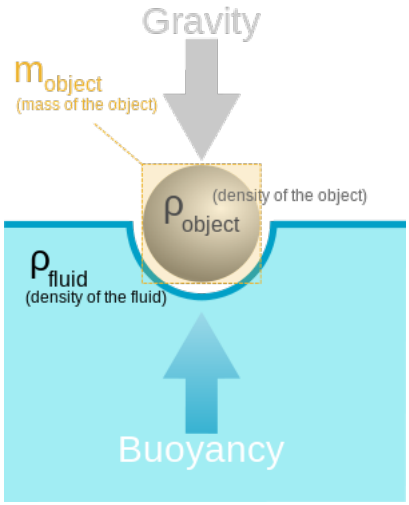


Figure 3.4: Buoyancy Force

3.4 Electronics

The electronics of the CubeSat are based upon the requirements of what the CubeSat needs to know. For the CubeSat to stabilise it needs to know at what orientation it is, an Inertial measurement Unit

1. Microcontroller
2. Inertial measurement unit (IMU)
3. Motor controller
4. Battery
5. Voltage regulator

1. Microcontroller A microcontroller forms the processing unit of the whole system. It processes all information that is given to it and sends signals to all other components connected to it. It forms an important component.

2. IMU An IMU(Inertia Measurement Unit) is a sensor that is equipped with a accelerometer that reads acceleration and a gyroscope rotation speed. This information is important for the CubeSat because the CubeSat needs to land on its bottom. Using this the roll,pitch and yaw of the CubeSat can be found. The Pitch is the change in horizontal plane from the CubeSats center, also the y-axis. Yaw is the change in direction of the z-axis. Roll is the vertical plane of the CubeSat's center also the x-axis. Also illustrated in the Figure (3.5) below. [4]

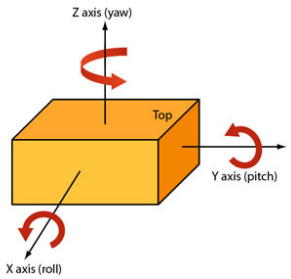


Figure 3.5: IMU orientation

3. Motor controller The H-Bridge is an electric circuit used to control DC motors.

4. Battery The cube needs a power source. This is delivered by a 9V batter for cost efficiency.

5. Voltage regulator Within the circuit there may exist different voltage levels, using voltage regulators this can be adjusted for the components.

Component	Amount	Usage
Atmel ATXMEGA256A3U	1	Microcontroller (Processing unit)
GY-88 IMU	1	Inertia measurement unit. Used to determine position and rotation.
XDRIVE 2430-65 DC motor	3	Used to drive the waterpumps.
L7805CV Voltage Regulator	1	Used to lower battery voltage to 5 volt.
LM317AHV Voltage Regulator	1	Used to lower 5 volt to 3.3 volt for the microcontroller.

Table 3.3: Components of CubeSat

3.5 Programming

The CubeSat will be coded in C. There are 2 important parts in the code. The first one is I^2C (Inter-Integrated Circuit). We are using the I^2C bus to communicate with our sensor.

The second is PWM(Pulse Width Modulation). PWM is used to control the speed of the motors. The PWM signal is generated with a duty cycle that is relative to the angle of the CubeSat.

Chapter 4

Functional Design & Implementation

From the research several components have been researched separately. In this chapter all the components will be combined to realise the functional design of the CubeSat. The CubeSat has mechanical components and electrical components which together form the CubeSat. These parts are intertwined which each other and will be discussed relative with each other. The basic functional design is shown in Figure 4.1. In the figure the several components that make up the CubeSat are shown with their relation to eachother. Each component will be discussed regarding their use and implementation.

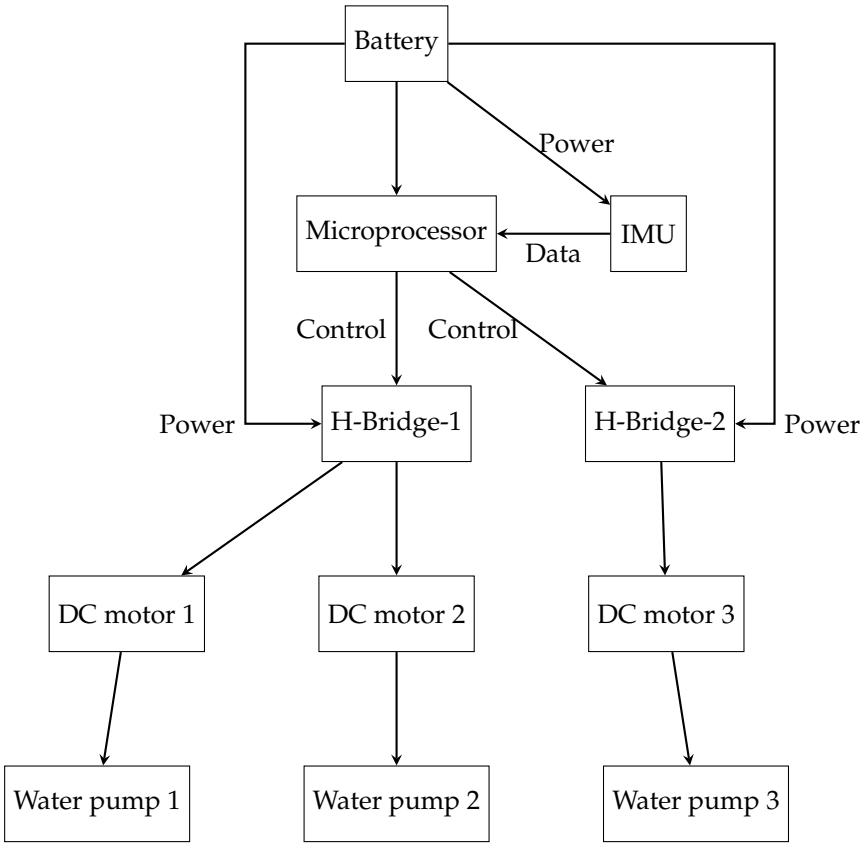


Figure 4.1: Functional scheme

An electrical wiring scheme of the CubeSat and the MM7150 made in Eagle can be found in Appendix II. The CubeSat design is shown in Appendix III

Microprocessor We chose the Atmel Xmega256A3U as microcontroller for our project, this microcontroller is relatively cheap and was recommened for our project due to the fact that we simply needed to adjust our 3 DC motors with readings from IMU. The atmel xmega256A3U supports PWM and I^2C, which allows to control the DC motors and connect with the IMU.

We implemented the PWM and I2C code using the skeleton code given by Wim Dolman and his book De

taal C en de Xmega [5]. Using this skeleton code we created cube_init() function that sets the pin to use for controlling the 3 DC motors (PORTE pin 0 to 7) and activating the pins for using I^2C (PORTD pin 0 to 4) [3]. Furthermore the i2c.h and i2c.c code on which the I^2C is based upon is shown in Appendix IV. Continiuing from line 74 code has been added to read or write data from and to a slave i2c device using the i2c device address and the internal adress on which you want to read from.

```
1  void cube_init(){
2    //I2C
3    i2c_init(&TWIE, TWI_BAUD(F_CPU, BAUD_100K));
4    PORTE.DIRSET     = PIN0_bm|PIN1_bm;                 // SDA 0 SCL 1
5    PORTE.PIN0CTRL   = PORT_OPC_WIREDANDPULL_gc;        // Pullup SDA
6    PORTE.PIN1CTRL   = PORT_OPC_WIREDANDPULL_gc;        // Pullup SCL
7    PMIC.CTRL |= PMIC_LOLVLEN_bm;
8
9    //PWM
10   PORTD.DIRSET = PIN0_bm; // PD0 PWM MOTOR1
11   PORTD.DIRSET = PIN1_bm; // PD2 PWM MOTOR2
12   PORTD.DIRSET = PIN2_bm; // PD4 PWM MOTOR3
13
14   TCD0.PER = PERIOD-1; // Define pwm-period
15   TCD0.CTRLA = TC_CLKSEL_DIV256_gc; // Prescaling 256
16   TCD0.CTRLB = TC0_CCAEN_bm | TC_WGMODE_SS_gc; // Single slope on
17
18   TCD0.CCA = PERIOD-(BASEDUTY*10); // Define dutycycle
19   TCD0.CCB = PERIOD-(BASEDUTY*10); // Define dutycycle
20   TCD0.CCC = PERIOD-(BASEDUTY*10); // Define dutycycle
21 }
```

CubeSat/main.c

IMU The GY-88 sensor module is a 10 Degrees of freedom sensor and has three chips, the MPU6050 (3-Axis Accelormenter & Gyro), HMC5883L (3-Axis magnetometer) and BMP085 for measuring the atmospheric pressure. We use the MPU6050 sensor for reading the accelerometer data and gyroscope data to determine our position. The use of magnetometer is avoided because it is unknown how much magnetic interference exists due to other components. Using I^2C we make connection with the MPU6050 and using the register map [6] we can acquire the data we need. The MPU6050 first needs to be initialised and this is done using the code below.

```
1  void mpu6050_init()
2  {
3    //set sleep disabled
4    mpu6050_setSleepDisabled();
5    //wake up delay needed sleep disabled
6    _delay_ms(100);
7    //set clock source of Z-axis of gyroscope as reference
8    i2c_writeBits(MPU6050_ADDRESS, PWR_MGMT_1, 2, 3, 0x03);
9    //set DLPF bandwidth to 256 Hz
10   i2c_writeBits(MPU6050_ADDRESS, CONFIG, 2, 3, 0x00);
11   //set sample rate
12   i2c_writeByte(MPU6050_ADDRESS,SMPLRT_DIV, 0x07); //8khz / (1 + 7) = 1000Hz
13   //set gyro range
14   i2c_writeBits(MPU6050_ADDRESS,GYRO_CONFIG, 4, 2, MPU6050_GYRO_FS);
15   //set accel range
16   i2c_writeBits(MPU6050_ADDRESS,ACCEL_CONFIG, 4, 2, MPU6050_ACCEL_FS);
17
18 }
```

CubeSat/GY88.c

In this function the MPU6050 sensor is configured and activated for use.

After the initialisation the values can be retrieved from the code with the following function:

```
void mpu6050_getRawData(float* axg, float* ayg, float* azg, float* gxds, float* gyds, float* gzds
    )
{

  int16_t ax = 0;
  int16_t ay = 0;
  int16_t az = 0;
  int16_t gx = 0;
  int16_t gy = 0;
  int16_t gz = 0;

  i2c_readBytes(MPU6050_ADDRESS,ACCEL_XOUT_H, 14, (uint8_t *)buffer);

    ax = (((int16_t)buffer[0]) << 8) | buffer[1];
    ay = (((int16_t)buffer[2]) << 8) | buffer[3];
    az = (((int16_t)buffer[4]) << 8) | buffer[5];
    gx = (((int16_t)buffer[8]) << 8) | buffer[9];
    gy = (((int16_t)buffer[10]) << 8) | buffer[11];
    gz = (((int16_t)buffer[12]) << 8) | buffer[13];

  *axg = (float)(ax)/MPU6050_AGAIN;
  *ayg = (float)(ay)/MPU6050_AGAIN;
  *azg = (float)(az)/MPU6050_AGAIN;
  *gxds = (float)(gx)/MPU6050_GGAIN;
  *gyds = (float)(gy)/MPU6050_GGAIN;
  *gzds = (float)(gz)/MPU6050_GGAIN;

}
```

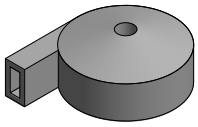
CubeSat/GY88.c

H-bridge The motors are controlled with a PWM signal sent by the H-bridge.The PWM signals duty-cycle is determined by the rotation of the CubeSat. For ex. if the cube is rotated 180 degrees in the y-axis the y-motor starts with a duty-cycle of 50 %.

```
void power_motor(){
  double upper = 0.0;
  double lower = 0.0;
  int duty_cycle = 0;

  if(rx > lower && rx < upper){
    duty_cycle = rx/3.6;
    TCD0.CCABUF = PERIOD-(duty_cycle*10); // Redefine dutycycle
  }

  if(ry > lower && ry < upper){
    duty_cycle = ry/3.6;
    TCD0.CCBBUF = PERIOD-(duty_cycle*10); // Redefine dutycycle
  }

  if(rz > lower && rz < upper){
    duty_cycle = rz/3.6;
    TCD0.CCCBUF = PERIOD-(duty_cycle*10); // Redefine dutycycle
  }
}
```

CubeSat/main.c

DC motors The DC motors are directly connected to the water pumps (shown below) and receive the signal from the H-bridge.



The following global control scheme can be defined for the functional design of the CubeSat. First the initialisation is started, then based upon the sensor values it is determined if the CubeSat is falling or not stabilised. The angles are calculated with the sensor values and if the CubeSat is not in the correct orientation a PWM signal is sent to the motor until the CubeSat is stabilised.

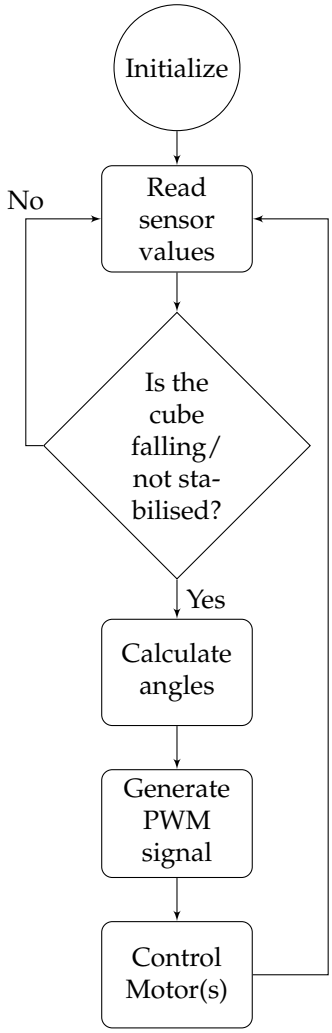


Figure 4.2: Control scheme

Chapter 5

Testing & Simulating

We could not test the full cube because we had problems while printing the cube and the water-pumps. Before the printed layer could cool down and dry the next layer was printed this caused the structure to bow and crack.

The IMU sensor provides raw data of the accelerometer and gyroscope readings, acquiring this wasn't too hard but to make use of this data there was need of writing Filtering and calibration algorithms and within the time span that was left this wasn't possible. This is due to the fact that the Raw Sensor readings can't be used to determine the orientation of the CubeSat
Our Minor coordinator, Patrick, had ordered the MM7150 IMU [7] which had built in filtering and calibration. This could provide greater ease of use due to the fact that the sensor delivers the correct readings of its position. But to use this sensor there was need of creating a circuit board, which had to be custom made in eagle due to the fact that there was no given library in which the components were defined. This introduced a new challenge and designing and soldering this was not easy, primarily due to the use of SMD components and making a package within eagle. Furthermore the code to communicate was not readily available and the code needed to be made in the short span of time. This requires intensive reading of the documentation and need of looking at the example code provided by MicroChip and adjusting this to use with our microcontroller.

Because of this we could not test the whole CubeSat by itself and only two of the most critical parts. All of the videos and code are available on `https://github.com/ikbenali/CubeSat.git`.

Chapter 6

Conclusion

We made a CubeSat which works in theory. All the sub components are working and while we can show and test these components one at a time we can't do a full test of the whole cubesat due to time constraints and lack of funding for the CubeSat. The concept of the water-pump works and we have video proof of this. The micro-controller can also read the sensor values and use these. This project does need more budget for the testing of different components and the whole CubeSat. We didn't realise that we chose the wrong sensor for this component. The GY-88 can work too but we need to do a lot more calculations while another sensor does all the calculations on its own.

Chapter 7

Recommendation

Coming back to the main question:
"Is it possible to simulation and reproduce the landing on the moon using a CubeSat in water?"
Yes it should be possible to simulate the landing of the CubeSat in water, this can be proven due to the fact that the pump mechanism works underwater and is able to drive at least one side of the CubeSat, furthermore with more experience of working with different pump designs a better efficiency can be reached. In case of a continiuation of this project the following things can be done to further improve the CubeSat:

Impeller and water-pump While our water-pump does work we think that the design can be changed to increase power and that allow the CubeSat to stabilise faster. This can be given due to the fact that the current dc motor slow down significantly due to the water resistance. Furthermore different impeller designs can be developed and compared to see which designs has the best performance.

3D printing We have printed the body of the CubeSat but this was only at 60% of the infill due to the fact that a 100% introduced warping of the structure. There is a lot of room improvements in the settings, also the type of printer that was available at school was too slow and not quick enough for the development and the amount of failed prints due to the printer warping with big structure was disadvantageous and not cost efficient for creating the CubeSat structure.
It would be wise to look at different fill densities and design methods which are easy for 3D printing development For ex. the fill density can be changed to strengthen and add weight to the cube.

IMU In a next iteration a better choice of sensor should be made, the sensor currently used had difficult documentation and was not as easy to use with atmel microcontrollers. Furthermore different sensor algorithms should be researched because it would allow the use of the current Sensor used and allow to keep the costs low for similar sensor.

Team size This project needs a few more team members. By having more members it would be easier to divide problems, such as the propulsion design and electronics,which will result in quicker and better results. It would be best to have people of multidisciplinary educations since this would bring more knowledge.

Bibliography

[1] Assad Anis. "Cold Gas Propulsion System - An Ideal Choice for Remote Sensing Small Satellites". In: (2012). URL: http://www.intechopen.com/books/remote-sensing-advanced-techniques-and-platforms/cold-gas-propulsion-system-an-ideal-choice-for-remote-sensing-small-satellites.

[2] Jonathan G. Fairman Carol Hodanbosi. *Buoyancy: Archimedes Principle*. 1996. URL: https://www.grc.nasa.gov/www/k-12/WindTunnel/Activities/buoy_Archimedes.html.

[3] Atmel Corporation. *ATxmega256A3U / ATxmega192A3U / ATxmega128A3U / ATxmega64A3U DATASHEET*.

[4] *Design*. URL: http://www.computersrwilde.com/Projects/hexacopter/design1.html.

[5] Wim Dolman. *De taal C en de Xmega*. ISBN: 978-90-484-3527-2.

[6] InvenSense Inc. *MPU-6000 and MPU-6050 Product Specification Revision 3.2*. 2011.

[7] Microchip. *MM7150 motion module*. 2015.

[8] A.H. de Ruiter, C. Damaren, and J.R. Forbes. *Spacecraft Dynamics and Control: An Introduction*. Wiley, 2012. ISBN: 9781118403327. URL: https://books.google.nl/books?id=mGSYHJIlDxEC.

[9] Brian Dunbar Sarah Loff. *Cubesat Overview*. URL: https://www.nasa.gov/mission_pages/cubesats/overview.

[10] The Engineering Toolbox. *Densities and molecular weights of some common gases - acetylene, air, methane, nitrogen, oxygen and others*. URL: http://www.engineeringtoolbox.com/gas-density-d_158.html.

[11] The Engineering Toolbox. *Density and specific weight of water at temperatures ranging 0 to 100 oC (32 to 212 oF) - in Imperial and SI Units*. URL: http://www.engineeringtoolbox.com/water-density-specific-weight-d_595.html.

[12] California Polytechnic State University. "CubeSat Design Specification Rev.13". In: (). URL: https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/56e9b62337013b6c063a655a/1458157095454/cds_rev13_final2.pdf.

[13] B.T.C. Zandbergen. *Cold gas systems*. URL: http://www.lr.tudelft.nl/en/organisation/departments/space-engineering/space-systems-engineering/expertise-areas/space-propulsion/propulsion-options/chemical-rockets/cold-gas/.

Appendices

Appendix I

CubeSat standards

The following are the requirements to send a CubeSat to a space mission.

I.1 General requirements

- The vessel is permitted up to 1.2 standard atmosphere (121590 Pascal) and will have a factor of safety no less then 4.
- Pyrotechnics and hazardous materials are not permitted.
- Chemical energy is allowed up to 100 Watt-Hour.
- The CubeSat is not allow to detach any materials at any time as this would become space debris.
- To prevent contamination of other spacecraft the following criteria have been placed for out-gassing.
  - The CubeSat shall not lose more than 1% of its total mass.
  - Collected Volatile Condensable Material shall be no more than 0.1%.
- The latest revision of the CubeSat design shall be used.

I.1.1 Exterior dimensions

A 1U CubeSat has to be 100 mm wide with a maximum error of 0.1 mm. The CubeSats height has to be 135 mm tall with the same error margin.

I.1.2 Mass

A 1U CubeSat has a maximum weight of 1.33 kg. The CubeSats center of gravity has to be located in a 2 cm sphere in the geometric center.

I.1.3 Materials

The CubeSat structure and rails have to be made from aluminium 7075 or 6061

I.2 Electrical requirements

The CubeSat is fully deactivated before the mission. The CubeSat shall power up all systems at start of the mission.

I.3 Operational requirements

The CubeSat can be shutdown remotely. All deploy-ables and transmitters will start after a set amount of time.

Appendix II

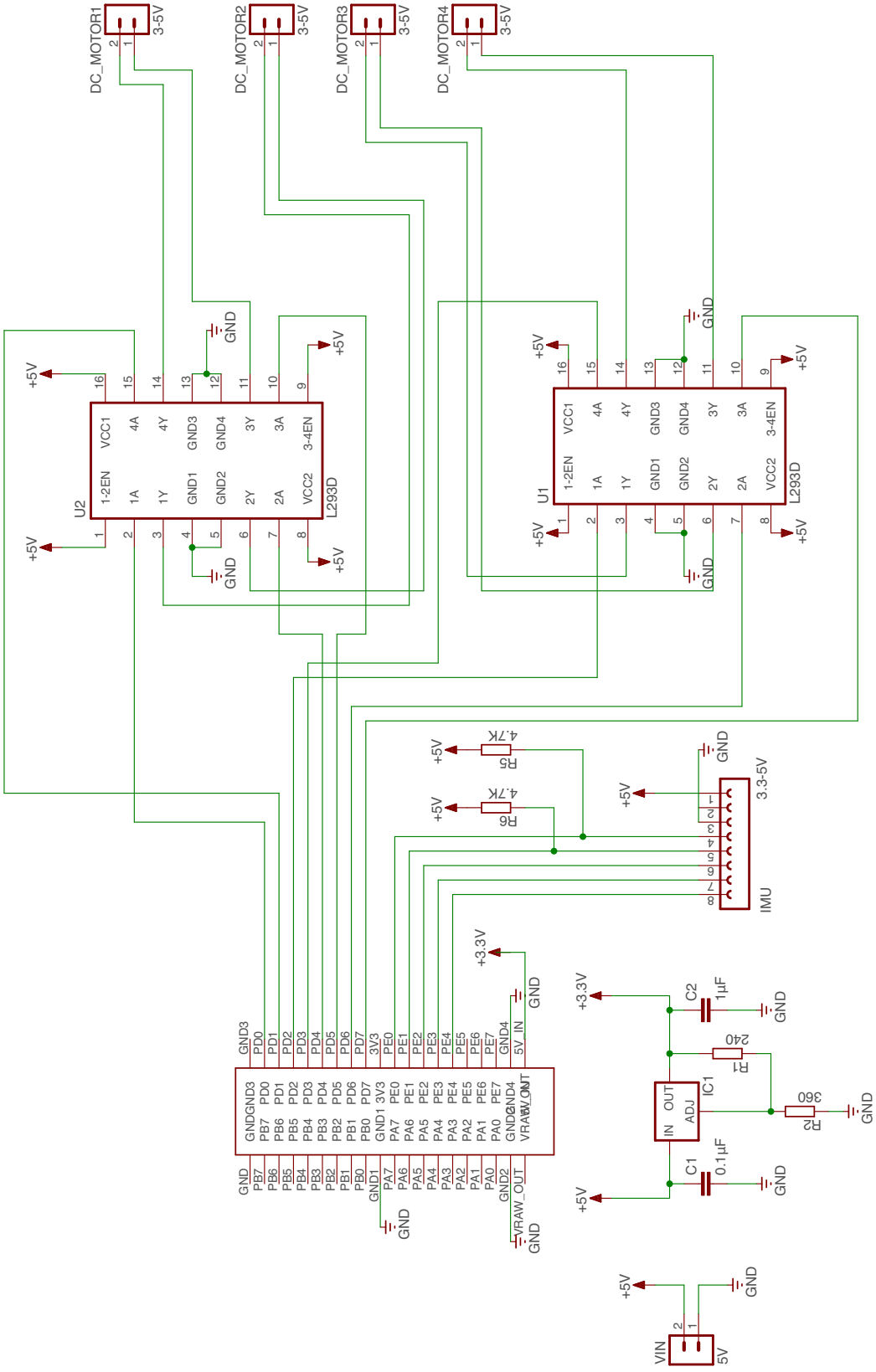
Electrical Scheme

II.1 Cubesat scheme

The CubeSat scheme consist of the following components:

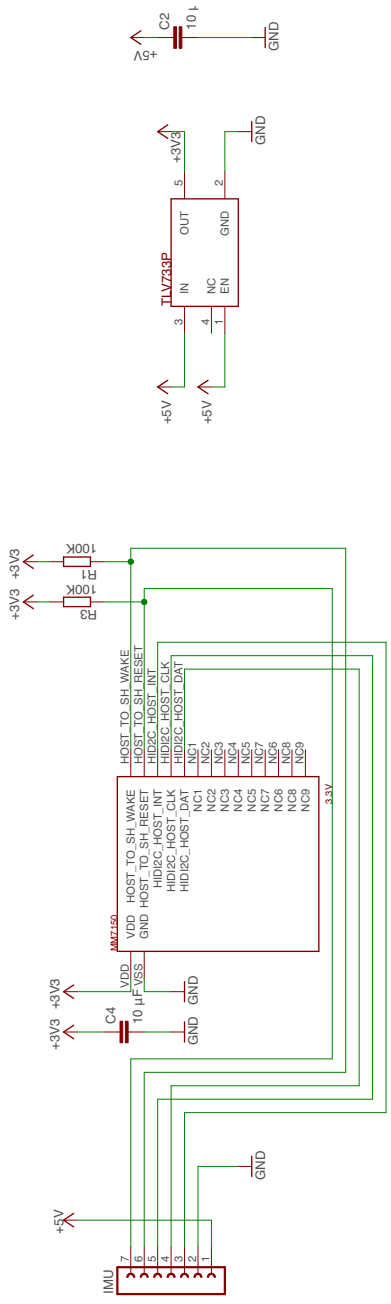
- XMega256A3U
- IMU
- L293DNE
- 4 DC motors
- Voltage Regulator
- Resistors
- Capacitors

Using a Female header the battery is inserted into the circuit, the battery has to deliver 5V and depending on how much the battery can deliver it needs to be converted to 5V before entering the circuit. Before the current enters the Microcontroller it is converted to 3.3V because the microcontroller can not handle the 5V delivered. The Microcontroller is connected to a Female Header with the Pull Up resistors of 4.7k Ohm on two lines, the SDA and SCL line of the IMU, to allow for stable transfer of data. The PORTD connections of the XMEGA connnect to the H-Bridges and the H-Bridges are connected to the DC motors, the CubeSat used 3 DC motors but a 4th one has been added to allow use of one additional motor/output. The scheme can be found on the next page.


DC_MOTOR1
DC_MOTOR2
DC_MOTOR3
DC_MOTOR4
3-5V
U2
U1
L293D
1-2EN
3-4EN
VCC1
VCC2
GND1
GND2
GND3
GND4
+5V
GND
R5
R6
4.7K
IMU
3.3-5V
+3.3V
C2
1µF
C1
0.1µF
R1
240
R2
360
IC1
IN
OUT
ADJ
VIN
5V

II.2 MM7150 scheme

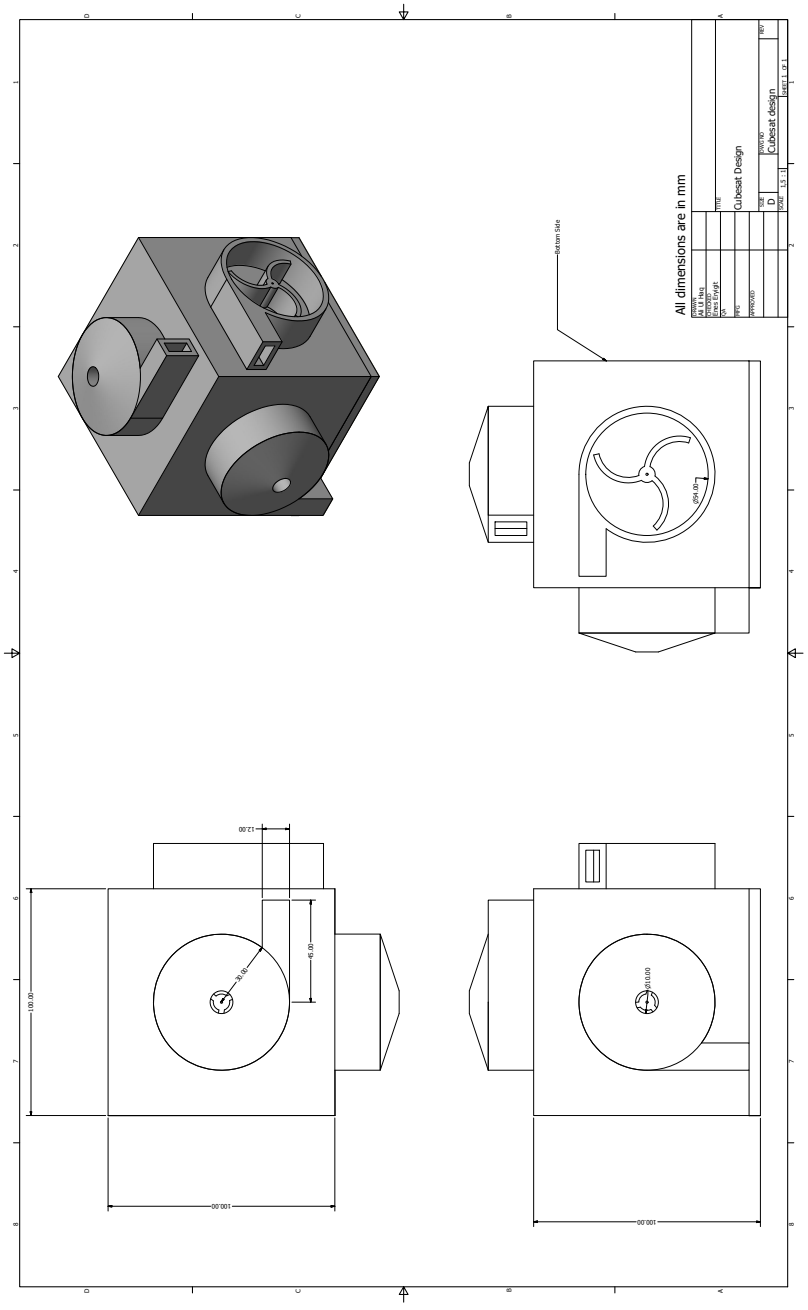
The MM7150 scheme is based on the connection of the microcontroller and MM7150. Due to the fact that the MM7150 scheme was created after the CubeSat scheme was designed it has different connection to the microcontroller. The SDA/SCL line do not exist on the same ports as it was on the previous MicroController. Furthermore the previous IMU (GY-88) could handle 5V whereas the MM7150 can not handle 5V. With the use of a voltage regulator the MM7150 can be used with the existing CubeSat scheme and by using different wiring the sensor can be connected.



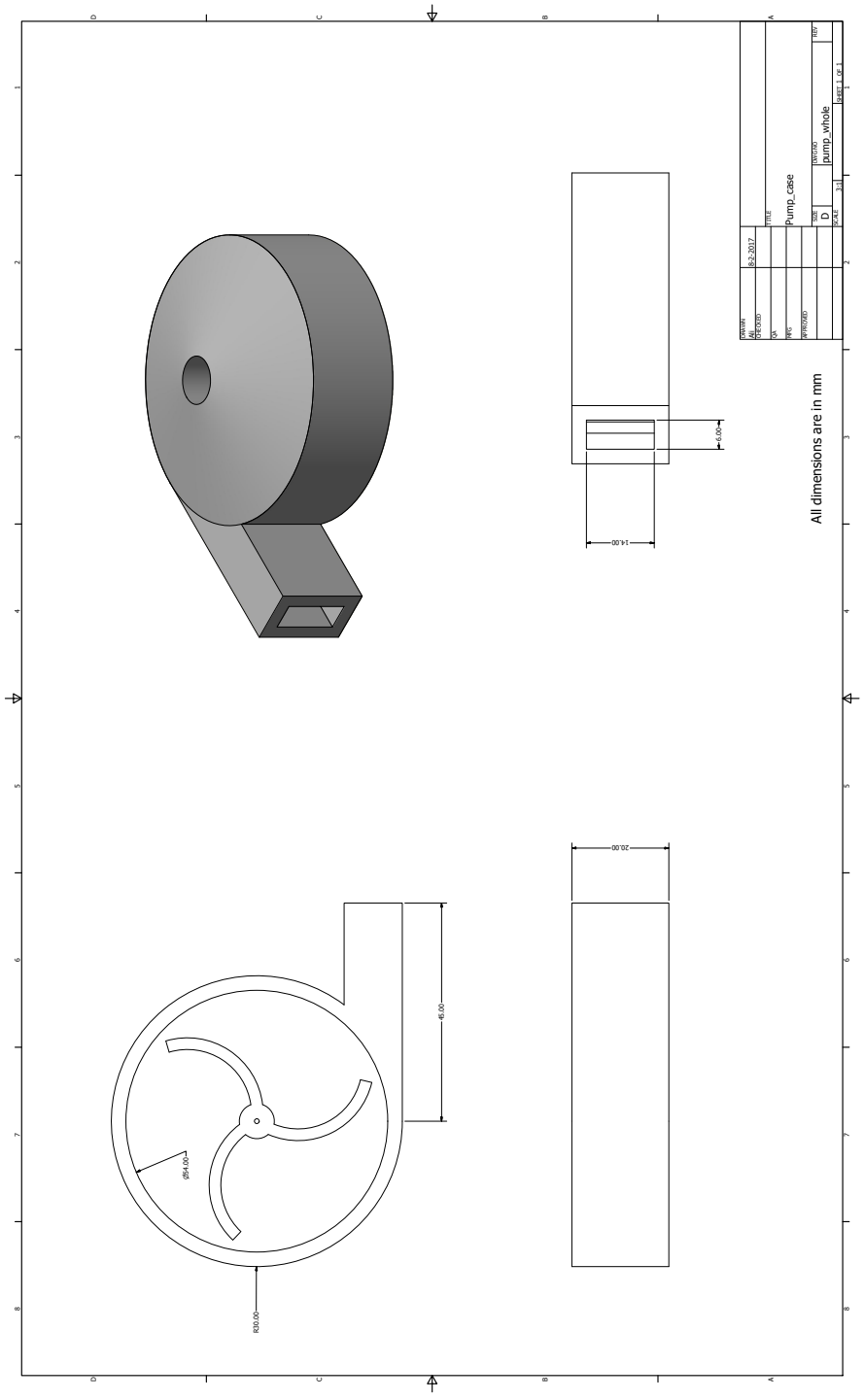
Appendix III

CubeSat design

III.1 Cubesat



III.2 Pump


Pump_case
pump_whole
All dimensions are in mm
6.00
14.00
20.00
45.00
ø54.00
R30.00

Appendix IV

I^2C

IV.1 i2c.h

```
#include <avr/io.h>
#include <util/delay.h>


#define TWI_BAUD(F_SYS, F_TWI)    ((F_SYS / (2 * F_TWI)) - 5)

#define I2C_ACK       0
#define I2C_NACK      1
#define I2C_READ      1
#define I2C_WRITE     0

#define I2C_STATUS_OK       0
#define I2C_STATUS_BUSY     1
#define I2C_STATUS_NO_ACK   2

void     i2c_init(TWI_t *twi, uint8_t baudRateRegisterSetting);
uint8_t i2c_start(TWI_t *twi, uint8_t address, uint8_t rw);
uint8_t i2c_restart(TWI_t *twi, uint8_t address, uint8_t rw);
void     i2c_stop(TWI_t *twi);
uint8_t i2c_write(TWI_t *twi, uint8_t data);
uint8_t i2c_read(TWI_t *twi, uint8_t ack);

int8_t mpu6050_readBytes(uint8_t regDevice, uint8_t regAddr, uint8_t length, uint8_t *data);
int8_t mpu6050_readByte(uint8_t regDevice, uint8_t regAddr, uint8_t *data);
void mpu6050_writeBytes(uint8_t regDevice, uint8_t regAddr, uint8_t length, uint8_t* data);
void mpu6050_writeByte(uint8_t regDevice, uint8_t regAddr, uint8_t data);
int8_t mpu6050_readBits(uint8_t regDevice, uint8_t regAddr, uint8_t bitStart, uint8_t length,
    uint8_t *data);
int8_t mpu6050_readBit(uint8_t regDevice, uint8_t regAddr, uint8_t bitNum, uint8_t *data);
void mpu6050_writeBits(uint8_t regDevice, uint8_t regAddr, uint8_t bitStart, uint8_t length,
    uint8_t data);
void mpu6050_writeBit(uint8_t regDevice, uint8_t regAddr, uint8_t bitNum, uint8_t data);
```

CubeSat/i2c.h

IV.2 i2c.c

```
#include "i2c.h"

void i2c_init(TWI_t *twi, uint8_t baudRateRegisterSetting)
{
  twi->MASTER.BAUD   = baudRateRegisterSetting;
  twi->MASTER.CTRLC  = 0;
  twi->MASTER.CTRLA  = TWI_MASTER_ENABLE_bm;
  twi->MASTER.STATUS = TWI_MASTER_BUSSTATE_IDLE_gc;
}

uint8_t i2c_start(TWI_t *twi, uint8_t address, uint8_t rw)
{
  if ( (twi->MASTER.STATUS & TWI_MASTER_BUSSTATE_gm) !=                    // if bus available
  TWI_MASTER_BUSSTATE_IDLE_gc ) return I2C_STATUS_BUSY; //
```

```
15   twi->MASTER.ADDR = (address << 1) | rw;                    // send slave
       address
16   while( ! (twi->MASTER.STATUS & (TWI_MASTER_WIF_bm << rw)) );  // wait until sent
17
18   if ( twi->MASTER.STATUS & TWI_MASTER_RXACK_bm ) {          // if no ack
19     twi->MASTER.CTRLC = TWI_MASTER_CMD_STOP_gc;
20     return I2C_STATUS_NO_ACK;
21   }
22
23   return I2C_STATUS_OK;
24 }
25
26 uint8_t i2c_restart(TWI_t *twi, uint8_t address, uint8_t rw)
27 {
28   twi->MASTER.ADDR = (address << 1) | rw;                    // send slave
       address
29   while( ! (twi->MASTER.STATUS & (TWI_MASTER_WIF_bm << rw)) );  // wait until sent
30
31   if ( twi->MASTER.STATUS & TWI_MASTER_RXACK_bm ) {          // if no ack
32     twi->MASTER.CTRLC = TWI_MASTER_CMD_STOP_gc;
33     return I2C_STATUS_NO_ACK;
34   }
35
36   return I2C_STATUS_OK;
37 }
38
39 void i2c_stop(TWI_t *twi)
40 {
41   twi->MASTER.CTRLC  = TWI_MASTER_CMD_STOP_gc;
42   twi->MASTER.STATUS = TWI_MASTER_BUSSTATE_IDLE_gc;
43 }
44
45 uint8_t i2c_write(TWI_t *twi, uint8_t data)
46 {
47   twi->MASTER.DATA = data;                                   // send data
48   while( ! (twi->MASTER.STATUS & TWI_MASTER_WIF_bm) );       // wait until sent
49
50   if ( twi->MASTER.STATUS & TWI_MASTER_RXACK_bm ) {          // if no ack
51     twi->MASTER.CTRLC = TWI_MASTER_CMD_STOP_gc;
52     return I2C_STATUS_NO_ACK;
53   }
54
55   return I2C_STATUS_OK;
56 }
57
58 uint8_t i2c_read(TWI_t *twi, uint8_t ack)
59 {
60   uint8_t data;
61
62   while( ! (twi->MASTER.STATUS & TWI_MASTER_RIF_bm) );       // wait until
       received
63   data = twi->MASTER.DATA;                                   // read data
64   twi->MASTER.CTRLC = ((ack==I2C_ACK) ? TWI_MASTER_CMD_RECVTRANS_gc :  // send ack (go on)
       or
65   TWI_MASTER_ACKACT_bm|TWI_MASTER_CMD_STOP_gc); //      nack (and stop)
66
67   if ( ack == I2C_NACK ) {
68     while( ! (twi->MASTER.STATUS & TWI_MASTER_BUSSTATE_IDLE_gc) );
69   }
70
71   return data;
72 }
73
74 /*
75  * read bytes from chip register
76  */
77 int8_t i2c_readBytes(uint8_t regDevice,uint8_t regAddr, uint8_t length, uint8_t *data) {
78   uint8_t i = 0;
```

```
79    int8_t count = 0;
80    if(length > 0) {
81      i2c_start(&TWIE, regDevice, I2C_WRITE);
82      i2c_write(&TWIE, regAddr);
83      _delay_us(10);
84      i2c_restart(&TWIE, regDevice, I2C_READ);
85      //read data
86      for(i=0; i<length; i++) {
87        count++;
88        if(i==length-1)
89          data[i] = i2c_read(&TWIE, I2C_NACK);
90        else
91          data[i] = i2c_read(&TWIE, I2C_ACK);
92      }
93      i2c_stop(&TWIE);
94    }
95    return count;
96  }
97
98  /*
99   * read 1 byte from chip register
100  */
101 int8_t i2c_readByte(uint8_t regDevice,uint8_t regAddr, uint8_t *data) {
102     return i2c_readBytes(regDevice,regAddr, 1, data);
103 }
104
105
106 /*
107  * write bytes to chip register
108  */
109 void i2c_writeBytes(uint8_t regDevice,uint8_t regAddr, uint8_t length, uint8_t* data) {
110   if(length > 0) {
111     //write data
112     i2c_start(&TWIE, regDevice, I2C_WRITE);
113     i2c_write(&TWIE, regAddr);
114     for (uint8_t i = 0; i < length; i++) {
115       i2c_write(&TWIE, (uint8_t) data[i]);
116     }
117     i2c_stop(&TWIE);
118   }
119 }
120
121 /*
122  * write 1 byte to chip register
123  */
124 void i2c_writeByte(uint8_t regDevice,uint8_t regAddr, uint8_t data) {
125     return i2c_writeBytes(regDevice,regAddr, 1, &data);
126 }
127
128 /*
129  * read bits from chip register
130  */
131 int8_t i2c_readBits(uint8_t regDevice,uint8_t regAddr, uint8_t bitStart, uint8_t length, uint8_t
      *data) {
132     // 01101001 read byte
133     // 76543210 bit numbers
134     //    xxx   args: bitStart=4, length=3
135     //    010   masked
136     //   -> 010 shifted
137     int8_t count = 0;
138     if(length > 0) {
139     uint8_t b;
140     if ((count = i2c_readByte(regDevice,regAddr, &b)) != 0) {
141       uint8_t mask = ((1 << length) - 1) << (bitStart - length + 1);
142       b &= mask;
143       b >>= (bitStart - length + 1);
144       *data = b;
145     }
```

```
146     }
147     return count;
148 }
149 
150 /*
151  * read 1 bit from chip register
152  */
153 int8_t i2c_readBit(uint8_t regDevice,uint8_t regAddr, uint8_t bitNum, uint8_t *data) {
154     uint8_t b;
155     uint8_t count = i2c_readByte(regDevice,regAddr, &b);
156     *data = b & (1 << bitNum);
157     return count;
158 }
159 
160 /*
161  * write bit/bits to chip register
162  */
163 void i2c_writeBits(uint8_t regDevice,uint8_t regAddr, uint8_t bitStart, uint8_t length, uint8_t
        data) {
164     //      010 value to write
165     // 76543210 bit numbers
166     //    xxx   args: bitStart=4, length=3
167     // 00011100 mask byte
168     // 10101111 original value (sample)
169     // 10100011 original & ~mask
170     // 10101011 masked | value
171   if(length > 0) {
172     uint8_t b = 0;
173     if (i2c_readByte(regDevice,regAddr, &b) != 0) { //get current data
174       uint8_t mask = ((1 << length) - 1) << (bitStart - length + 1);
175       data <<= (bitStart - length + 1); // shift data into correct position
176       data &= mask; // zero all non-important bits in data
177       b &= ~(mask); // zero all important bits in existing byte
178       b |= data; // combine data with existing byte
179       i2c_writeByte(regDevice,regAddr, b);
180     }
181   }
182 }
183 
184 /*
185  * write one bit to chip register
186  */
187 void i2c_writeBit(uint8_t regDevice,uint8_t regAddr, uint8_t bitNum, uint8_t data) {
188     uint8_t b;
189     i2c_readByte(regDevice,regAddr, &b);
190     b = (data != 0) ? (b | (1 << bitNum)) : (b & ~(1 << bitNum));
191     i2c_writeByte(regDevice,regAddr, b);
192 }
```

CubeSat/i2c.c