CUBITRON

Felix Strand **Brage Wiseth**

Introduction

In this project, we aim to design and build a self-balancing cube using reaction wheels, a type of control mechanism commonly used in spacecraft for attitude control. The goal is to create a compact, dynamically stable system capable of maintaining its equilibrium in three-dimensional space without external support. The cube, with its simple geometric structure, presents unique challenges in terms of balance and orientation control, making it an excellent platform for exploring the principles of feedback control systems and mechanical design.

Reaction wheels, which function by spinning at varying speeds to produce torques that counteract disturbances, will serve as the core actuators for balancing the cube. By carefully coordinating the speeds of these wheels, the system will be able to counteract any rotational forces, allowing the cube to maintain stability in any orientation.

This project will involve multiple stages, including the design and fabrication of the cube, the selection and integration of reaction wheels, the development of sensors to track its orientation, and the implementation of a control algorithm that adjusts the wheel speeds in real-time to correct any deviations. The project not only challenges our understanding of dynamic systems and control theory but also has the potential to inform future applications in robotics, aerospace, and precision engineering, where balance and stability are crucial.

Ultimately, the self-balancing cube will serve as a proof-of-concept for demonstrating the potential of reaction wheels in balancing and controlling small-scale systems, offering a hands-on opportunity to delve into the fascinating intersection of mechanical engineering, robotics, and control theory.

First Principles

How can we balance with just a set of spinning wheels that don't even touch the ground? Well the same way you and I would balance on a slack line or on a fence post, We pivot our arms around our torso kinda like how a plane would turn its wings when banking. How this makes us balance is not at all obvoius, but it has something to do with angular momentum and torques, considering that we are rotating our body. Another hint is that by extending our arms further out or even using a rod to artifitially extend our arms we can greatly enhance our ability to balance. By doing so we create torque, we can use this exact theqnique with flywheels only instead of it beeing two arms we essentially have infinite arms stacked in a circle.

Circuitry

The internal logic and the physical output of our system needs to be joined together by circuitry; a way to translate our calculations for balance onto the motors themselves. Our circuitry has three main responsibilites; input -> calculate -> output. Our input will be in form of sensordata indicating our cubes current angle, and a wanted state (do we want to balance? on which edge? so on). Our calculations will be about transforming our current input angles into a quantative action for the motors to perform. The output will in turn be the adjustment of our motors to achieve balance. High level, this will require accelerometers and gyroscopes as our input, a microcontroller to perform the calculations, and motors to be adjusted for output. All this combined can be reasonably packed into a few circuits that contain a few sensors, a microcontroller, and motordriver capabililities.

This section of our documentation aims to provide context and reasoning behind our main circuit choices, as well as give a general overview of all components, their role in Cubitron specifically, and how they are interconnected to achieve a cohesive circuit with results.

Components

To gain context of our implementation, this subsection will outline every integrated circuit we use, what their general purpose in a project is, and more specifically what their purpose is for Cubitron.

Microcontroller

Our microcontroller will be the ATSAME53J18A-MF, which is has a 32-bit ARM Cortex-4 processor. Our main reasons for this microcontroller is the ceiling for calculations and peripherals. We expect to make reasonably heavy calculations in the context of an embedded system, which the 32 bit system with an in built Floating Point Unit is excellent for. Additionally, we need to connect two sensors and six motors, which makes the pool of peripherals we need to support quite large. We also have personal experience with the usage of these microcontrollers, with similar problem statements, and were satisfied with the physical capabilities of our microcontroller.

The microcontroller has the most central and complicated role of the whole circuitry. It will be the component responsible of combing the three main stages of our program runtime. The pipeline of Input -> Calculate -> Output has the microcontroller as a major component in all of them. Our sensors will both be sending their data through an I2C connection to our microcontroller. They will have an assigned interrupt pin each, which opens for having our sensor data be as close to real time as possible. From this sensor data, we need to calculate our current position in space, with regard to our 3D orientation, and calculate again how far this is from our intended position, and what motor actions we need to perform to get there. Lastly, our output stage will consist of sending that data to our TMC controllers, and adjusting our motors physically.

As you can imagine, our microcontroller is hooked up to a of external components in our circuitry, which is quite nice to have an overview over. Below is a table of the names of all external connections in the perspective of our microcontroller schematics, the endpoint they are connected to, the pin and pin type it is connected to, and a short description of its purpose:

Name	**Endpoint**	**Pin**	**Pin Type**	**Function**
SWDIO	Programmer	PA31	Digital	Dataline for programming of our microcontroller
SWCLK	Programmer	PA30	Digital	Clock signal for programming
ERR_LED	LED	PA12	Digital	LED indicating system error
PROCESS_LED	LED	PA13	Digital	LED indicating processing
IMU1 INT	IMU1	PA05	Digital	IMU1 data rdy interrupt
IMU1 SCL	IMU1	PA06	Digital	IMU1 Serial Clock for data transfer
IMU1 SDA	IMU1	PA07	Digital	IMU1 Serial data line
IMU2 INT	IMU2	PA23	Digital	IMU2 data rdy interrupt
IMU2 SCL	IMU2	PA24	Digital	IMU2 Serial Clock for data transfer
IMU2 SDA	IMU2	PA25	Digital	IMU2 Serial data line
OLED_SDA	OLED	PA08	Digital	OLED screen Serial Data Line
OLED_SCL	OLED	PA09	Digital	OLED screen Serial CLK
S1_SIG	Servo 1	PA17	Digital	Brake Servo 1 control signal

Name	Endpoint	Pin	Pin Type	Function
S2_SIG	Servo 2	PA18	Digital	Brake Servo 2 control signal
S3_SIG	Servo 3	PA19	Digital	Brake Servo 3 control signal
DB_TX	DB_HEADER	PA22	Digital	UART TX for debug purposes
DB_RX	DB_HEADER	PA22	Digital	UART RX for debug purposes

Programmer

We actually have two individual microcontrollers on this board, but the second one is intended for programming our main microcontroller. If you looked at the pin table for the ATSAME53J18A-MF, we have two entries with the endpoint "Programmer", which refers to this component. The programmer equips a 32-bit ARM Cortex M0+ processor, and was chosen because of the close relation to the microcontroller above, and being able to act as a bridge for programming and debugging the Cortex-M4, with the firmware CMSIS-DAP. This explains the pins between the programmer and our main Cortex-M4.

CMSIS-DAP (Cortex Microcontroller Interface Standard) is an ARM standard for the DAP (Debug Access Port) that provides an USB interface to allow debugging and programming ARM Cortex-M microcontrollers. Our Cortex-M0 will have an USB-C interface through the physical connector of our microcontroller, which will communicate with the device we are flashing from (probably our computers). Furthermore, we can use DAP functionality for debugging and flashing our controller. We can program the flash memory through our USB interface, debug (stop execution, set breapoints, view memory, etc…) our programs.

Read more about CMSIS-DAP:

https://arm-software.github.io/CMSIS_5/DAP/html/index.html

Name	Endpoint	Pin	Pin Type	Function
SWDIO	Microcontroller	PA30	Digital	Dataline for programming of our microcontroller
SWCLK	Microcontroller	PA31	Digital	Clock signal for programming
D+	Microcontroller	PA25	Digital	USB Interface

Name	Endpoint	Pin	Pin Type	Function
D-	Microcontroller	PA24	Digital	USB Interface

Motorcontroller

This is the bread and butter of our system, and allows us to make a precise and awesome motordriver circuit. The TMC4671 allows for control of a BLDC motor, and will be tripled up for the control of three BLDC motors in total. The benefit of this is being allowed to control the motors completely separated, with no interference between the three. The TMC4761 supports Field Oriented Control, which allows for very precise and energy efficient moving of the motors.

Inertial Measurement Unit

We will in total have three separate circuits, wherein two of them we be dedicated to host a single IMU (Inertial Measurement Unit) each. The rationale for this is the positions we will mount the sensors, to acquire the information about our orientation in order to stabilize the system at a given angle. One sensor will be mounted on the corner which will be facing the ground during operation, and the other sensor will be on the corner straight above the corner facing the ground, so they form a straight line. We can then deduce the orientation and angle of the cube, in such a way that we know our cube is balancing straight if $\theta = \frac{\pi}{2}$ with respect to the ground, and $\delta\theta \approx 0$.

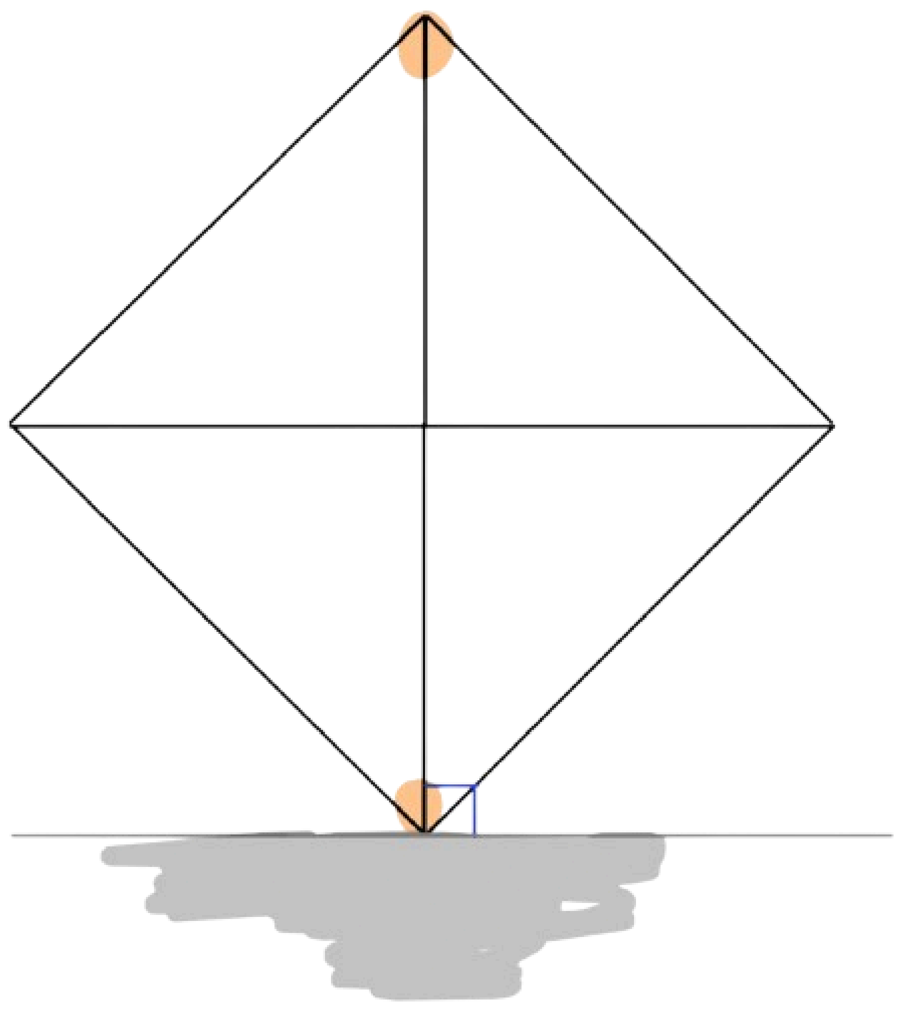

Figure 1: Orange indicates IMU positions. Blue highlights the angle of our IMU-line with respect to the ground.