Technical Documentation Avionics Bay
Elpis MK IIb

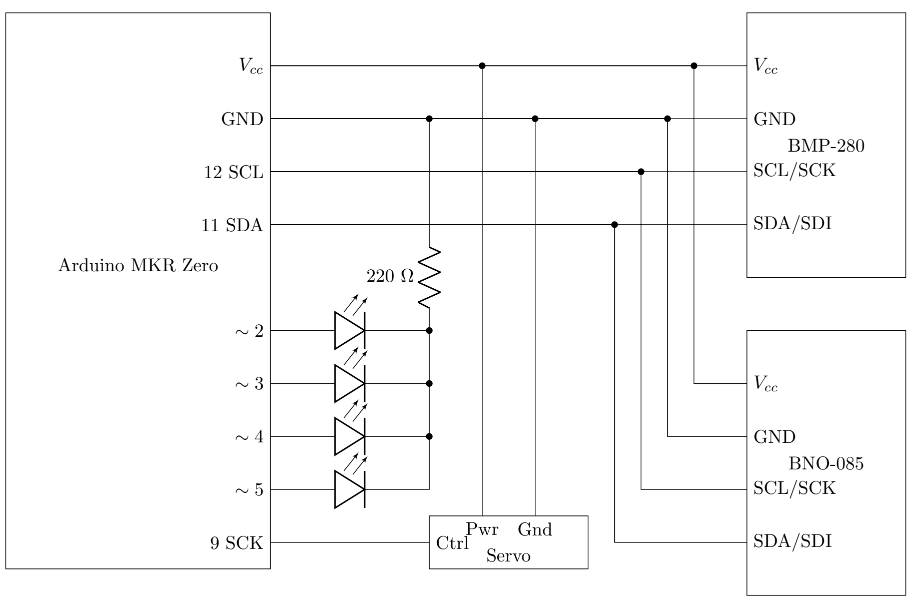
Mytikas

Hardware

Components

Microcontroller	Arduino MKR Zero
Barometer	BMP 280
Accelerometer	BNO 085
Servo	

Circuit diagram



Pinmap

Printed name	Compiler name	Variable name	Use
V_{cc}	n/a	n/a	Power supply voltage
GND	n/a	n/a	Ground
12 SCL	n/a	n/a	Clock signal for I^2C
11 SDA	n/a	n/a	Data signal for I^2C
9 SCK	9	SERVO_PIN	Signal for parachute servo
~ 2	2	LED_PIN	Status LED
~ 3	3	ERROR_LED_PIN	Error LED
~ 4	4	LAUNCH_LED_PIN	Launch LED
~ 5	5	CARD_LED_PIN	Card LED

Software

0.1 Accelerometer class

Public Interface

The Accelerometer class's public interface has a constructor and a `getData()` method. The contructor initializes the accelerometer, and sets the appropriate values. the getData method takes a pointer to a telemetry struct as an argument. It reads values from the accelerometer (Bno085), processes them and writes them to the `telemetry` instance.

Private parts

Constructor The constructor begins by initializing most of the members. It then repeatedly tries to start the BNO, and lights the Error LED during failure. It then repeatedly attempts to set the desired reports. If it were to fail, the Error LED is lit until success.

getData() `getData()` starts out by checking if the BNO was reset. If it was, then the reports are set again. It also logs the reset in the `telemetry` instance. Then it atempts to read the reports in a while loop. After that it checks if a read value is zero. This happens sometimes, we don't know why. If it is zero, then the last nonzero value is put in its place. This is why the next step is writing the `trot` and `tacc` values to `acc` and `rot` members. `acc` and `rot` are then written to the `telemetry` instance. A vector `racc` is then created by rotating `acc` by `rot`. It is also written to the `telemetry` instance.

0.2 Vector Types

There are currently two vector types, quaternions(`Quat`) and three dimensional vectors(`Vec3`).

Quaternion

A quaternion is a hypercomplex number. This means it has four components; one real, and three imaginary(i, j, k).

$$Q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \tag{1}$$

As for normal complex number there are special arithmetic rules for the imaginary parts. Addition is done component-wise, multiplication is done using a Hamilton product(see muliply). Quaternions are often used to represent a rotation in 3d-space, and that is what they are doing in this codebase. A rotation of θ degrees around the axis (x,y,z) would look like

$$Q = cos\frac{\theta}{2} + xsin\frac{\theta}{2}\mathbf{i} + ysin\frac{\theta}{2}\mathbf{j} + zsin\frac{\theta}{2}\mathbf{k} \tag{2}$$

as a quaternion. A rotation quaternion should have a magnitude 1, which is accomplished when the rotation axis is normalized.

Constructors There are two constructors, one without arguments, which returns a zero `quat`, and one with four arguments. The four arguments are one for each component of the vector, and sets the members to these values.

print() There is a print function implemented for `quat`s. The first argument is the name of the vector. This will be printed out before the values. The second argument determines whether or not to put a linebreak at the end. The call `Quat().print("Rotation", false)` will print
`Rotation: r: 0.0, i: 0.0, j: 0.0, k: 0.0.`

invert() This functions inverts the `quat`. This is the same as taking the complex conjugate of the number, that is, negating all imaginary parts:

$$\bar{Q} = a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k} \tag{3}$$

mulitply() This returns the Hamilton product of two quaternions. This product is not commutative, that is $Q_1 * Q_2 \neq Q_2 * Q_1$. This functions takes an argument `q2` and multiplies it onto the object from the right, meaning that `q1.mulipIy(q2)` is mathematically eqiavalent to $q1 * q2$, and not $q2 * q1$. Mathematically, the product looks like this:

$$\begin{aligned}
Q_1 &= a_1 + b_1\mathbf{i} + c_1\mathbf{j} + d_1\mathbf{k} \\
Q_2 &= a_2 + b_2\mathbf{i} + c_2\mathbf{j} + d_2\mathbf{k} \\
Q_1 * Q_2 &= a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2 \\
&+(a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)\mathbf{i} \\
&+(a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)\mathbf{j} \\
&+(a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)\mathbf{k}
\end{aligned}$$

This represents the rotation Q_1 followed by the rotation Q_2, if both Q_1 and Q_2 are rotation Quaternions.

Vector 3

Constructors `Vec3` has two constructors, an empty one which returns the zero vector, and one with three arguments for the x, y and z components respectively.

print() Prints out the `Vec3` to Serial. The first argument is the name of the vector and will be printed before the values. The second one detmermines whether to break the line at the end. The call
`Vec3().print("Acceleration", false)` prints
`Acceleration: x: 0.0, y: 0.0, z: 0.0`

rotate() The rotate function rotates the `Vec3` object with a quaternion. Mathematically, rotating the vector V with Q would look like this:

$$Q_V = 0.0 + V_x\mathbf{i} + V_y\mathbf{j} + V_z\mathbf{k} \tag{4}$$
$$Q_R = Q * Q_V * \bar{Q} \tag{5}$$
$$\tag{6}$$

Here the imaginary components of Q_R are the x, y and z components of the rotated V. The multiplication sign means the Hamilton product.