Kon-tiki final technical report: COMPASS

About the project

the goal of the project is to give us an acurate reading of the rockst state eg velocity, position, orientation, and acceleration. this is so that when the time is right we can deploy the parachute and land the rocket safely without tearing it apart. it wil also give us a better understanding of the rocket and how it behaves in the air. and more easily recover the rocket We do this by using a combination of sensors and a microcontroller to process the data and send it to the ground station as well as store it on the rocket for redundancy and flight data analysis.

Summary

The project is a success, we have a working prototype that has been tested in the field and in the lab. the major takeaway is that the system works well for altitude estimation and orientation. but x and y velocity and position is prone to drift. The reason for this is the lack of absolute positioning and velocity sensors like GPS and airspeed sensors. The orientation and altitude states have such sensors and are therefore more accurate. Despite this, the system does a good job of giving a rough estimate of the velocity and position. With so called “dead reckoning” we can estimate the position and velocity without the need of telecommunication that GPS needs.

Testing

[kalman unit tests]
[pressure sensor tests]
[magnetometer calibration tests]
[imu visualisation tests]
[drone flight tests]

Development

As the project was based on previous projects the specs were already set in stone. and they did not change much during the development of the project. however the last iteration swapped the magnetometer for a more accurate one. but the overall design and specs stayed the same.

Timeline

Modeling

The navigation system is based on a state-space representation of the system. The state of the system is defined as the orientation, velocity, and altitude of the rocket. The dynamics of the system are described by a set of differential equations. The sensors are used to measure the state of the system, and the Kalman filter is used to estimate the state of the system. To estimate the state of the rocket, we must first build a model of the rocket’s dynamics and the measurement models of the sensors. The dynamics of the rocket can be described by a set of differential equations. The state of the rocket is defined as a vector of variables that describe the orientation, velocity, and altitude of the rocket.

The general form of a dynamical system can be stated as

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{w}_d$$
$$\hat{\boldsymbol{y}} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{w}_n$$

where $\boldsymbol{x}$ is the state vector, $\boldsymbol{u}$ is the control vector, $\boldsymbol{w}_d$ is the system disturbance, $\boldsymbol{y}$ is the measurement vector, $\boldsymbol{w}_n$ is the measurement noise, $\boldsymbol{f}$ is the dynamics of the system. For linear systems we can use linear algebra to express our system

$$\begin{aligned}\dot{\boldsymbol{x}} &= \boldsymbol{F}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{w}_d \\ \hat{\boldsymbol{y}} &= \boldsymbol{H}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u} + \boldsymbol{w}_n\end{aligned}$$

we can also descreteize our system (which is what we must do in order to use our computer algorithms)

$$\begin{aligned}\boldsymbol{x}_{t+1} &= \boldsymbol{f}_d(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w}_d \\ \hat{\boldsymbol{y}}_t &= \boldsymbol{h}_d(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w}_n \\ &\text{in the linear case} \\ \boldsymbol{x}_{t+1} &= \boldsymbol{F}_d\boldsymbol{x}_t + \boldsymbol{B}_d\boldsymbol{u}_t + \boldsymbol{w}_d \\ \hat{\boldsymbol{y}}_t &= \boldsymbol{H}_d\boldsymbol{x}_t + \boldsymbol{D}_d\boldsymbol{u}_t + \boldsymbol{w}_n\end{aligned} \tag{1}$$

in our case the vector $\boldsymbol{u}$ is 0 since we have no controll over the rocket as far as the navigation system is concerned. The state vector $\boldsymbol{x}$ is defined as

$$\boldsymbol{x} = \begin{pmatrix} \text{orientation} \in \mathbb{R}^4 \\ \text{velocity} \in \mathbb{R}^3 \\ \text{altitude} \in \mathbb{R} \end{pmatrix}$$

Coordinate systems

Before we can define the state of the rocket, we must agree upon the coordinate systems. There are a some popular ones such as the NED (North-East-Down) and ENU (East-North-Up). These coordinate systems define the choice of basis vectors (the directions of the axes).

We use the ENU coordinate system where the x-axis points east, the y-axis points north, and the z-axis points up.

Orientation and quaternions

The orientation of the rocket is represented by a quaternion. A quaternion can be thought of as an extenstion of complex numbers. It can be used to represent rotations and orientations in 3D space, depending on the context. The quaternion is defined as

$$\boldsymbol{q} = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$$

where a, b, c, d are real numbers and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are the imaginary units. It can be efficiently represented on a computer as a 4D vector.

$$\boldsymbol{q} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

Without diving too deep into the math, the quaternion can be used to represent a rotation

$$\hat{\boldsymbol{v}} = \boldsymbol{q}\boldsymbol{v}\boldsymbol{q}^{-1}$$

$\hat{\boldsymbol{v}}$ is the result of rotating $\boldsymbol{v}$ by $\boldsymbol{q}$. where $\boldsymbol{v}$ and $\hat{\boldsymbol{v}}$ is 3D vectors expressed as a quaternion with the scalar part being 0.

$$\boldsymbol{v} = \begin{pmatrix} 0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} \qquad \hat{\boldsymbol{v}} = \begin{pmatrix} 0 \\ \hat{v}_1 \\ \hat{v}_2 \\ \hat{v}_3 \end{pmatrix}$$

In this example we have used the complex conjuate and the quaternion multiplication operations.

Quaternion conjugate

For a unit quaternion the inverse is equal to the conjugate $\boldsymbol{q}^{-1} = \bar{\boldsymbol{q}}$

$$\bar{\boldsymbol{q}} = \begin{pmatrix} a \\ -b \\ -c \\ -d \end{pmatrix}$$

Quaterinion multiplication

$$\boldsymbol{q}_1\boldsymbol{q}_2 = \begin{pmatrix} a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2 \\ a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2 \\ a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2 \\ a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2 \end{pmatrix}$$

This operation is in general not commutative. so the order of the quaternions matter $\boldsymbol{q}_1\boldsymbol{q}_2 \neq \boldsymbol{q}_2\boldsymbol{q}_1$. But it is associative meaning that $(\boldsymbol{q}_1\boldsymbol{q}_2)\boldsymbol{q}_3 = \boldsymbol{q}_1(\boldsymbol{q}_2\boldsymbol{q}_3)$.

Quaterinion normalization

The magnitude of a quaternion is defined as

$$||\boldsymbol{q}|| = \sqrt{\boldsymbol{q}\bar{\boldsymbol{q}}} = \sqrt{\bar{\boldsymbol{q}}\boldsymbol{q}} = \sqrt{a^2 + b^2 + c^2 + d^2}$$

we want to *normalize* the quaternion so that it has a magnitude of 1, and consequently

$$\boldsymbol{q}\bar{\boldsymbol{q}} = ||\boldsymbol{q}||^2 = 1$$

we can normalize the quaternion by dividing it by its magnitude

$$\boldsymbol{q}_u = \frac{\boldsymbol{q}}{||\boldsymbol{q}||}$$

Quaterinion kinematics

The derivative of a quaternion is defined as

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q}\boldsymbol{\omega}$$

where $\boldsymbol{\omega}$ is the angular velocity of our system. By using a gyro sensor, we can obtain $\omega = \begin{pmatrix} 0 \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$ With this equation, we can integrate the angular velocity to get the orientation of the rocket.

$$\boldsymbol{q}_{t+1} = \boldsymbol{q}_t + \frac{1}{2}\boldsymbol{q}\boldsymbol{\omega}\,\mathrm{d}t$$

Velocity and altitude

The velocity of the rocket is represented as a 3D vector

$$\boldsymbol{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

the velocity is obtained by integrating the acceleration of the rocket. However, we want the velocity in respect to the world frame, and thus we cannot directly integrate each component of the acceleration in a decoupled manner. What if the rocket is tilted say along the x-axis? In this case, the acceleration along the z-axis will also have a component along the x-axis in the world frame. The rocket frame is rotated with respect to the world frame. To combat this, we must find a mapping or a translation if you will, between the rocket frame and the world frame. Lucky for us, we have the quaternion that can be used to rotate the acceleration vector to the world frame. So the velocity is obtained by integrating the acceleration in the world frame.

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t + \boldsymbol{q}\boldsymbol{a}\bar{\boldsymbol{q}}\,\mathrm{d}t$$

Finally, the altitude is a scalar donoting the height of the rocket above the sea level in meters. We can simply integrate the velocity along the z-axis to get the altitude.

$$z_{t+1} = z_t + v_z\,\mathrm{d}t$$

Dynamics

As we have seen, the dynamics is governed by the following equations

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = \begin{pmatrix} \boldsymbol{q}_{t+1} = \boldsymbol{q}_t + \frac{1}{2}\boldsymbol{q}\boldsymbol{\omega}\,\mathrm{d}t \\ \boldsymbol{v}_{t+1} = \boldsymbol{v}_t + \boldsymbol{q}\boldsymbol{a}\bar{\boldsymbol{q}}\,\mathrm{d}t \\ z_{t+1} = z_t + v_z\,\mathrm{d}t \end{pmatrix}$$

we can think of the input $\boldsymbol{u}$ as beeing our $\boldsymbol{\omega}$ and $\boldsymbol{a}$ from the sensors.

Orientation dynamics

Lets first consider the orientation part. This equation is in fact a linear equation in the state variable $\boldsymbol{q}$, However, We want to express the equation in a matrix-vector multiplication form

$$\boldsymbol{q}_{t+1} = \boldsymbol{F}_{\boldsymbol{q}}\boldsymbol{q}_t$$

Where $\boldsymbol{F}_{\boldsymbol{q}}$ is a matrix. We can achive this by expanding our quaternion kinematics equation to get

$$\begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{pmatrix} + \begin{pmatrix} q_1\omega_0 - q_2\omega_x - q_3\omega_y - q_4\omega_z \\ q_1\omega_x + q_2\omega_0 + q_3\omega_z - q_4\omega_y \\ q_1\omega_y - q_2\omega_z + q_3\omega_0 + q_4\omega_x \\ q_1\omega_z + q_2\omega_y - q_3\omega_x + q_4\omega_0 \end{pmatrix} \frac{1}{2}\,\mathrm{d}t$$

and we can see that this can be expressed as a matrix-vector multiplication $\boldsymbol{F}_{\boldsymbol{q}}\boldsymbol{q}$ where the matrix $\boldsymbol{F}_{\boldsymbol{q}}$ is

$$\boldsymbol{F}_{\boldsymbol{q}} = \begin{pmatrix} 1 & -\frac{1}{2}\omega_x\,\mathrm{d}t & -\frac{1}{2}\omega_y\,\mathrm{d}t & -\frac{1}{2}\omega_z\,\mathrm{d}t \\ \frac{1}{2}\omega_x\,\mathrm{d}t & 1 & \frac{1}{2}\omega_z\,\mathrm{d}t & -\frac{1}{2}\omega_y\,\mathrm{d}t \\ \frac{1}{2}\omega_y\,\mathrm{d}t & -\frac{1}{2}\omega_z\,\mathrm{d}t & 1 & \frac{1}{2}\omega_x\,\mathrm{d}t \\ \frac{1}{2}\omega_z\,\mathrm{d}t & \frac{1}{2}\omega_y\,\mathrm{d}t & -\frac{1}{2}\omega_x\,\mathrm{d}t & 1 \end{pmatrix}$$

Velocity dynamics

As with the orientaion dynamics we want to express the equations by a matrix-vector product. However the velocity dynamics is not linear since the quaternion components are coupled in a non-linear way. We get expressions like q_1^2 and $q_1 q_2$ which are non-linear. In order to get the matrix we want we can linearize, by taking the first derivative of our velocity equation we get the jacobian matrix

$$\boldsymbol{F}_{vq} = \begin{pmatrix} 2\big(q_1 a_x + q_3 a_z - q_4 a_y\big)\,\mathrm{d}t & 2\big(q_2 a_x + q_3 a_y + q_4 a_z\big)\,\mathrm{d}t & 2\big(q_1 a_z - q_3 a_x + q_2 a_y\big)\,\mathrm{d}t & 2\big(q_4 a_x - q_1 a_y + q_2 a_z\big)\,\mathrm{d}t \\ 2\big(q_1 a_y + q_4 a_x - q_2 a_z\big)\,\mathrm{d}t & 2\big(q_3 a_x - q_2 a_y - q_1 a_z\big)\,\mathrm{d}t & 2\big(q_3 a_y + q_2 a_x + q_4 a_z\big)\,\mathrm{d}t & 2\big(q_3 a_z - q_4 a_y + q_1 a_x\big)\,\mathrm{d}t \\ 2\big(q_1 a_z + q_2 a_y - q_3 a_x\big)\,\mathrm{d}t & 2\big(q_4 a_x - q_2 a_z + q_1 a_y\big)\,\mathrm{d}t & 2\big(q_4 a_y - q_3 a_z - q_1 a_x\big)\,\mathrm{d}t & 2\big(q_4 a_z + q_2 a_x + q_3 a_y\big)\,\mathrm{d}t \end{pmatrix}$$

$$\boldsymbol{F}_{vv} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Position dynamics

This one is simple, to express the position dynamics in a matrix-vector multiplication form we can simply write

$$\boldsymbol{F}_a = (0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; \mathrm{d}t \;\; 1)$$

$$z_{t+1} = \boldsymbol{F}_a \begin{pmatrix} \vdots \\ v_z \\ z_t \end{pmatrix}$$

Putting it all together we get the dynamics matrix $\boldsymbol{F}$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \boldsymbol{F} = \begin{pmatrix} \boldsymbol{F}_q & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{F}_{vq} & \boldsymbol{F}_{vv} & \boldsymbol{0} \\ \cdots & \boldsymbol{F}_a & \cdots \end{pmatrix}$$

Measurement models

Systems have different degrees of observability. Some states are observable from the sensors, while others are not. The orientation of the rocket can be measured by sensors such as magnetometers, sun trackers and star trackers. The altitude can be directly measured by barometers, and both altitude and velocity can be measured by GPS. Though GPS is not used in this system. With the sensors we have, we still need a way to translate between the sensor readings and our state model. That is what measurement models are for. They describe how the state of the system is related to the sensor readings.

Orientation measurement

To measure the orientation of the rocket, we use a magnetometer. The magnetometer measures the magnetic field of the earth. The magnetic field of the earth is known and is going nowhere (anytime soon), so it can be used as a reference. We need a reading of the magnetic field when the rocket is stationary, and the orientation is in it's initial state.

$$\text{reference} \coloneqq \boldsymbol{\psi}_0$$

The translation that relates the magnetic field to the orientation of the rocket is

$$\hat{\boldsymbol{\psi}} = \boldsymbol{q}\boldsymbol{\psi}_0\bar{\boldsymbol{q}}$$

In other terms, the system thinks the output of the magnetometer will be $\hat{\boldsymbol{\psi}}$.
Anytime we want to measure the orientation of the rocket, we take a reading of the magnetic field $\boldsymbol{\psi}$ and compare it to the prediction.

$$\hat{\boldsymbol{\psi}} = \boldsymbol{q}\boldsymbol{\psi}_0\bar{\boldsymbol{q}}$$
$$\varepsilon = \boldsymbol{\psi} - \hat{\boldsymbol{\psi}}$$

The error ε is the difference between the measured orientation and the predicted orientation. The error is then used to correct the orientation of the rocket. We will see how this is done in the Kalman filter section.

Similar to the jacobian for the velocity dynamics, we can linearize the orientation measurement model to get the jacobian matrix

$$\boldsymbol{H}_q = \begin{pmatrix} 2\left(q_1\psi_x + q_3\psi_z - q_4\psi_y\right) & 2\left(q_2\psi_x + q_3\psi_y + q_4\psi_z\right) & 2\left(q_1\psi_z - q_3\psi_x + q_2\psi_y\right) & 2\left(q_4\psi_x - q_1\psi_y + q_2\psi_z\right) \\ 2\left(q_1\psi_y + q_4\psi_x - q_2\psi_z\right) & 2\left(q_3\psi_x - q_2\psi_y - q_1\psi_z\right) & 2\left(q_3\psi_y + q_2\psi_x + q_4\psi_z\right) & 2\left(q_3\psi_z - q_4\psi_y + q_1\psi_x\right) \\ 2\left(q_1\psi_z + q_2\psi_y - q_3\psi_x\right) & 2\left(q_4\psi_x - q_2\psi_z + q_1\psi_y\right) & 2\left(q_4\psi_y - q_3\psi_z - q_1\psi_x\right) & 2\left(q_4\psi_z + q_2\psi_x + q_3\psi_y\right) \end{pmatrix}$$

Altitude measurement

The altitude can be measured by a barometer. The barometer measures the pressure of the air. The pressure of the air decreases with altitude.

$$h = h_0 + l_0\left(\frac{p_0}{p}\right)^{g_0\frac{m_0}{r_0}l_0}$$

As we lack a GPS, we cannot measure the velocity of the rocket directly. The only way to update the velocity is via the dynamics of the system. This is a common problem in navigation systems and is called the observability problem. The velocity is not observable from the sensors, and we must rely on the dynamics of the system

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}} = \boldsymbol{H} = \begin{pmatrix} \boldsymbol{H}_q & \boldsymbol{0} \\ \boldsymbol{0} & h \end{pmatrix}$$

State estimation with Kalman filter

With the state-space representation of the system and the linearized dynamics, we can use a Kalman filter to estimate the state of the rocket. For pure linear systems, the Kalman filter is optimal and can be used to estimate the state of the system given noisy measurements. However, the system is not purely linear, so we must use an extended Kalman filter which is not optimal but does a good job of estimating the state of the system nonetheless. The extended Kalman filter is identical to the Kalman filter except for the linearization step. The Kalman filter functions kinda like a feedback controller. It takes the measurements

and the estimated state and corrects the estimated state based on the error between the measurements. Recall the system dynamics (1)

$$\boldsymbol{x}_{t+1} = \boldsymbol{F}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{w}_d$$
$$\hat{\boldsymbol{y}}_t = \boldsymbol{H}\boldsymbol{x}_t + \boldsymbol{D}\boldsymbol{u}_t + \boldsymbol{w}_n$$

We can add the feedback loop to the system dynamics to get the Kalman filter

$$\hat{\boldsymbol{x}}_{t+1} = \boldsymbol{F}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{K}(\boldsymbol{y} - \hat{\boldsymbol{y}})$$

where $\boldsymbol{y}$ is the actual measurement and $\hat{\boldsymbol{y}}$ is the predicted measurement from our measurement model. Below is a very barebones illustration of the Kalman filter

The choice of the gain K is not arbitrary, it is chosen to minimize the cost function

$$J = E[\boldsymbol{x} - \boldsymbol{x}_{\hat{}}]^T Q E[\boldsymbol{x} - \boldsymbol{x}_{\hat{}}] + E[\boldsymbol{y} - \boldsymbol{y}_{\hat{}}]^T R E[\boldsymbol{y} - \boldsymbol{y}_{\hat{}}]$$

Recall the state-space representation of the system equation (1) where we have system disturbances $\boldsymbol{w}_d$ and measurement noise $\boldsymbol{w}_n$. Now a very important assumption that the Kalman filter bases its optimality on is that the noise is Gaussian.

which is the expected value of the error between the true state and the estimated state. This i a quadratic cost function that is minimized by the Kalman filter. By solving the Riccati equation, we can find the optimal Kalman gain $\boldsymbol{K}$

Hardware

The PCB is designed in KiCad and is a 4-layer board. The board is powered by a 3.3V LDO regulator. In general, the PCB design is centered around the ARM Cortex M4 microcontroller and the sensors. The sensors are connected to the microcontroller via a shared SPI bus, so the microcontroller has to select the sensor it wants to communicate with by setting the chip select pin of the sensor. The only exception is the magnetometer which is connected via I2C directly to the ISM330DHCX which have an I2C bus and DSP functionality. Most ICs have a decoupling capacitors as recommended by the manufacturer.

ARM Cortex M4

The ARM Cortex M4 is a 32-bit microcontroller that is used to run the navigation system. The processing power of the microcontroller is sufficient to run the navigation algorithms and communicate with the sensors. The peripherals used by the microcontroller are the SPI, USART, Timer, and SD card. It also provides a interuppt vector table that we use for the state machine. The sensor drivers is only a matter of reading and writing to the registers of the sensors with SPI

ISM330DHCX

[visualisation testing]

Barometer

[Pressure testing]

Magnetometer

[magnetometer calibration testing]

Firmware

The codebase uses the harmony framework to interface with the peripherals of the microcontroller. Harmony is a code generator that generates code based on the configuration of the peripherals. It provides a high-level API to interface with the peripherals of the microcontroller.

The main loop of the firmware is a state machine that runs the navigation algorithms. The state machine is implemented as a switch-case statement that switches between the different states of the system. Some states are triggered by interrupts from the sensors, while others are triggered by a timer. The state machine is responsible for reading the sensor data, running the navigation algorithms, and sending the data to the ground station.

API

The API of the firmware is quite limited. The firmware is designed to be a standalone system that runs the navigation algorithms and communicates with the sensors. The only way to interact with the firmware is through the SERCOM USART interface. The firmware sends the data to the ground station in a binary format. The data is sent in packets that contain the sensor data and the estimated state of the rocket.

Testing

The most useful tests are the SERCOM inteface in debug mode. By connecting the system to a computer we can print useful information about the state of the system. The tests are run in the lab and in the field. The project also comes with visualizations of the data that can be used to analyze the performance of the system. and unit tests that can be run independently of the microcontroller. sensor specific tests are also run to ensure that the sensors are working as expected.

Integration

This sevtion is about how the system is integrated into the rocket. How the system is powered and how it communicates with the ground station and other systems. We innitially used FRAM but switched to usart for simplicity and higher data rates as the FRAM was slow. It is connected directly to the recovery system and the RF module. The physical position of the system is together with the rest of the electronics, ideally in the center of mass of the rocket. Shielding is also preffered to limit sensor noise and interference.

Possible improvements

air speed sensor gps better magnetometer maybe a sun/star/earth tracker

What did we learn?

Compare the system that was before to the system we have now. What was improved What was not improved What was actively worse

Literature and resources