Chapter 4

Absolute Orientation and Dead Reckoning

As seen in Chapter **??**, having an accurate orientation estimate of the iSBL array is crucial for getting the position estimate of the array. Without it, the system is unable to transform the relative position estimate to the global/test frame. This chapter details how that orientation estimate is formed. Additionally, the concept of "dead reckoning" is introduced and its integration into the iSBL-SF algorithm is explained.

The first section of this chapter covers the various methods of orientation estimation, compares them to the method chosen for this thesis, and explores some implementations in related works. Next, the choice of IMU for this thesis is discussed, and its software implementation (interfacing with STM32, initialization, and calibration) is explained. The chosen orientation estimation method for this thesis, the Madgwick filter, receives its own section: it covers the theory of the Madgwick filter (including an explanation of quaternions) and the code implementation of the filter. Dead reckoning is introduced and its software execution is shown. Finally, both the orientation estimation system and the dead reckoning system are validated experimentally.

4.1 Background and Previous Works

This section is split into two subsections: the first covers the history of orientation estimation and the development of MEMS IMUs; the second compares different sensor fusion algorithms for converting MEMS IMU readings into an absolute orientation estimate, and mentions some implementations of these filters.

4.1.1 Introduction to IMUs and Orientation Estimation

Estimating the absolute orientation of a system (generally, the orientation relative to the global frame) is a common problem in both robotics and marine navigation. Early ocean navigators, such as the Phoenecians and the Polynesians, used celestial and environmental navigation techniques to sail from place to place. It took nearly 2000 years of development for compasses to become reliable and commonplace in ships, and another 500 years before gyrocompasses (electrically-powered gyroscopes that maintain a certain orientation and can be aligned to point towards north) were invented [1]. In the past 100 years, orientation and heading estimation technology has advanced at an incredible rate; mechanical gyroscopes have improved and can be shrunk down to fit inside of commercial planes, spacecraft, and missile systems [2]. While quite costly, modern mechanical and ring-laser gyroscopes can provide extremely accurate orientation estimates for mission-critical systems [1].

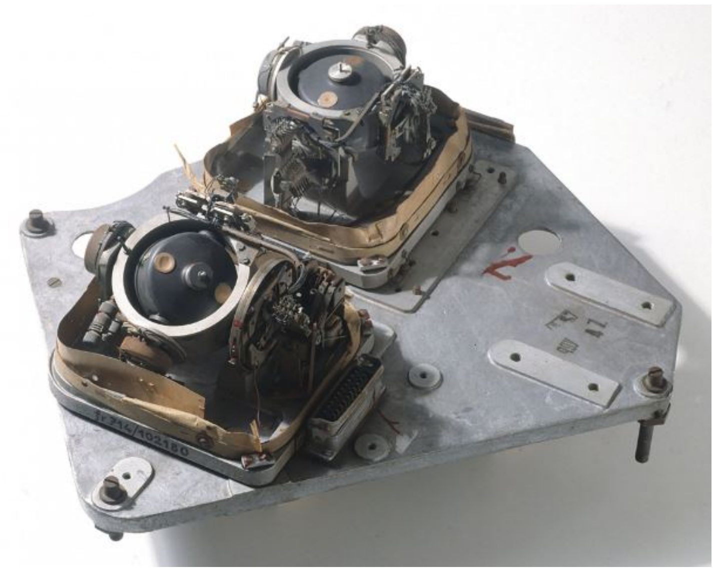


Figure 4.1: V2 missile gyroscope [2]

With the development of semiconductor technology, a new inertial navigation technology was created: MEMS IMUs. Micro-electro-mechanical systems (MEMS) are small devices made using semiconductor-fabrication technology. Inertial measurement units (IMUs) are systems that combine accelerometers (which measure linear acceleration and gravity), gyroscopes (which measure angular velocity), and sometimes magnetometers (which measure magnetic field strength, like that of the Earth's). IMUs are used in many modern aircraft, boat, and vehicle systems for attitude and heading reference. These IMU components can be manufactured as MEMS devices and are often packaged as integrated circuits, which can be as small as a ladybug (see Figure 4.2) [3].

These MEMS IMUs are not as accurate as their macro-sized equivalents, but have considerable benefits over them. First, they can be orders of magnitude less expensive than traditional IMUs; the IMU purchased for this thesis cost $25, while a non-MEMS IMU would often be in the thousands of dollars. They are also much easier to set up and maintain, not having any macro-scale moving components. Lastly, they are commonly used in modern, small-scale robotics systems and have gained a large consumer market. MEMS IMUs come in different performance levels with varying degrees of accuracy and applications (hobbyist and consumer, industrial, high-end tactical), providing a good range of options for different price points and needs [3]. See Figure 4.3 for a comparison of these performance levels.

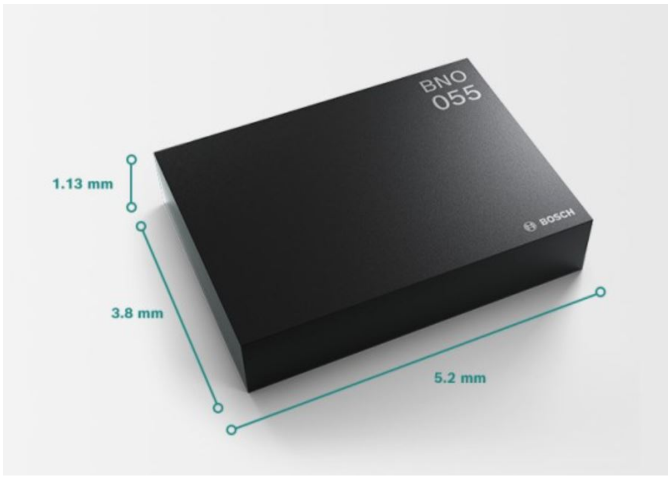
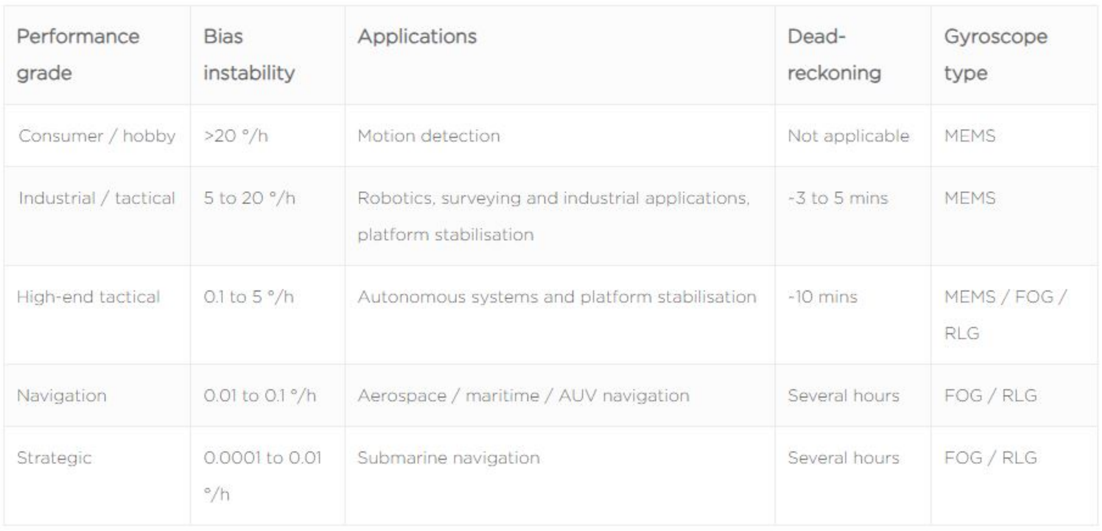


Figure 4.2: BNO055 IMU integrated circuit chip size [4]



Performance grade	Bias instability	Applications	Dead-reckoning	Gyroscope type
Consumer / hobby	>20 °/h	Motion detection	Not applicable	MEMS
Industrial / tactical	5 to 20 °/h	Robotics, surveying and industrial applications, platform stabilisation	~3 to 5 mins	MEMS
High-end tactical	0.1 to 5 °/h	Autonomous systems and platform stabilisation	~10 mins	MEMS / FOG / RLG
Navigation	0.01 to 0.1 °/h	Aerospace / maritime / AUV navigation	Several hours	FOG / RLG
Strategic	0.0001 to 0.01 °/h	Submarine navigation	Several hours	FOG / RLG

Figure 4.3: IMU performance level comparison [3]

The components of MEMS IMUs do not automatically generate an orientation estimate; an algorithm must be applied to convert the raw acceleration, angular velocity, and (sometimes) magnetic field strength readings into a usable orientation estimate relative to the global frame. This method, combining measurements from multiple unique sensors to obtain an estimate of some parameter (like orientation) is called sensor fusion. High-end MEMS IMUs tend to perform this sensor fusion for the user, while most low-cost MEMS IMUs (like the one in this thesis) require the user to compute the orientation estimation themselves. Some low-cost MEMS IMUs like the Bosch BNO055 do come with sensor fusion algorithms built in [4], but these techniques are generally proprietary and this approach is less common for hobbyist-level MEMS IMUs.

For the remainder of this paper, it will be assumed that the IMUs used will include magnetometers, and that they are MEMS IMUs. An IMU that contains a magnetometer may be called a MARG (magnetic, angular rate, and gravity) [5], a MIMU (magnetic inertial measurement unit) [6], or just an IMU [3]. For simplicity, this thesis will use this last approach. Whenever IMU is mentioned in this paper, it should be assumed that it contains a 3-axis accelerometer, gyroscope, and magnetometer.

4.1.2 IMU Sensor Fusion Algorithms

The basic idea of any IMU sensor fusion algorithm is combining the long-term accuracy of accelerometers and magnetometers with the short-term accuracy of gyroscopes. When stationary, 3-axis accelerometers can detect the gravitational field of Earth and the strength along each axis to determine where "down" is. Similarly, when stationary and isolated from other magnetic fields, a 3-axis magnetometer can detect the magnetic field of Earth and use it to determine where magnetic north is; it should be noted that "true north" and "magnetic north" do not point in the same direction and that the meaning of "down" can vary at different longitudes and latitudes, two things that must be taken into consideration for advanced / "mission-critical" systems [3].

In an ideal world, these two sensors could provide the 3D orientation relative to the Earth frame for any object that they are attached to. However, real-world systems are often not perfectly stationary, and robotic systems in particular tend to have stray magnetic fields produced by large electric motors and other circuitry in the system. A gyroscope can be used to account for short-term movements by measuring the angular velocity of the system; these measurements are primarily used when the system is rotating or accelerating, where the accelerometer and magnetometer readings become noisy. A gyroscope alone cannot be used for accurate orientation estimation; even if the initial orientation of the system was known perfectly, any miniscule error would propagate over time and produce a wildly-drifting orientation estimate. This problem will return in Section 4.5.

For low-cost MEMS IMUs, there are a few sensor fusion algorithms that are commonly used. For these examples, the orientation will be represented by the quaternion $\mathbf{q}$ - quaternion representation will be discussed in Section 4.4.1. The most basic is a complementary filter, where a weighted average combines the orientation estimate given by the accelerometer and magnetometer with the change in orientation given by the gyroscope [6]. An example of a complementary filter is shown below in Equation 4.2.

$$\mathbf{q}_n = (1-\gamma)\mathbf{q}_{acc/mag} + \gamma(\mathbf{q}_{n-1} + \Delta t \times \mathbf{q}_{gyro}) \tag{4.1}$$

A Mahony filter, invented by Robert Mahony et al. in 2010, attempts to minimize the error between the gyroscope orientation estimate and the accelerometer/magnetometer orientation estimate. It implements a proportional and integral feedback controller to minimize this error, and uses two tunable gains [7]. Figure 4.4 shows the algorithm for a Mahony filter.

$$\begin{aligned}
s_{\hat{a}_{q,t}} &= \hat{q}_{t-1}^{-1} \otimes E_{a_{q,t}} \otimes \hat{q}_{t-1} \\
s_{\hat{m}_{q,t}} &= \hat{q}_{t-1}^{-1} \otimes E_{m_{q,t}} \otimes \hat{q}_{t-1} \\
\\
s_{w_{mes,t}} &= [s_{a_t} \times s_{\hat{a}_t}] + [s_{m_t} \times s_{\hat{m}_t}] \\
s_{\dot{\hat{w}}_{b,t}} &= -k_i s_{w_{mes,t}} \\
s_{\hat{w}_{r,q,t}} &= s_{w_{q,t}} - [0 \quad s_{\hat{w}_{b,t}}] + [0 \quad k_p s_{w_{mes,t}}] \\
\\
\dot{\hat{q}} &= \tfrac{1}{2}\hat{q}_{t-1} \otimes s_{\hat{w}_{r,q,t}}
\end{aligned}$$

Figure 4.4: Mahony filter algorithm [8]

A Madgwick filter, invented by Sebastian Madgwick in 2010, is a gradient descent-based orientation estimation filter. It formulates the problem as an optimization task, attempting to minimize the error between the current (overall) orientation estimate and the accelerometer/magnetometer orientation estimate. It compensates for gyroscope and magnetometer bias/drift and only uses one tuning parameter [5]. Section 4.4.2 covers the theory in more depth.

An extended Kalman filter is possibly the most common algorithm used for orientation estimation of IMUs. Kalman filters are explained in depth in Chapter **??**; an extended Kalman filter is a modified version that works with non-linear systems (standard Kalman filters assume a linear state-space model). One particular implementation of a USBL system, mentioned in Section **??**, used an extended Kalman filter to merge raw IMU data with acoustic position estimates. This approach by Morgado, Oliveira, and Silvestre from the Technical University of Lisbon, Portugal gave very promising results - close to a 15% improvement level compared to loosely-coupled acoustic positioning systems [9] (like the one in this thesis). This tightly-coupled approach was considered for this thesis, but was deemed out-of-scope. For future implementations, a more tightly-coupled approach would be recommended.

The Mahony, Madgwick, and extended Kalman filters are very comparable in accuracy; generally, extended Kalman filters are the most accurate out of the three, but come at a relatively high computational cost compared to Mahony and

Madgwick filters [5] [6]. These two filters are also specifically formulated and optimized to work with IMUs, while the extended Kalman filter is a more general framework. See Figure 4.5 for an accuracy comparison of the three filters in a quadrotor implementation, and the associated time for 6041 filter updates of each algorithm in Figure 4.6 [8]. See Figure 4.7 for the orientation estimate accuracy of a Madgwick filter, versus Figure 4.8 for the accuracy of a Mahony filter, both in a pedestrian gait-measuring implementation [6].

	X	Y	Z	Norm
Madgwick	3.4677	3.3482	12.0775	13.0039
Mahony	4.1204	4.7005	9.06644	11.0107
EKF	1.4088	1.2055	13.1648	13.2948

Figure 4.5: Accuracy comparison for Madgwick, Mahony, and extended Kalman filters in quadrotor implementation [8]

	Execution time (seconds)
Madgwick	0.2080
Mahony	0.1782
EKF	0.2895

Figure 4.6: Computation time comparison for 6401 filter updates [8]

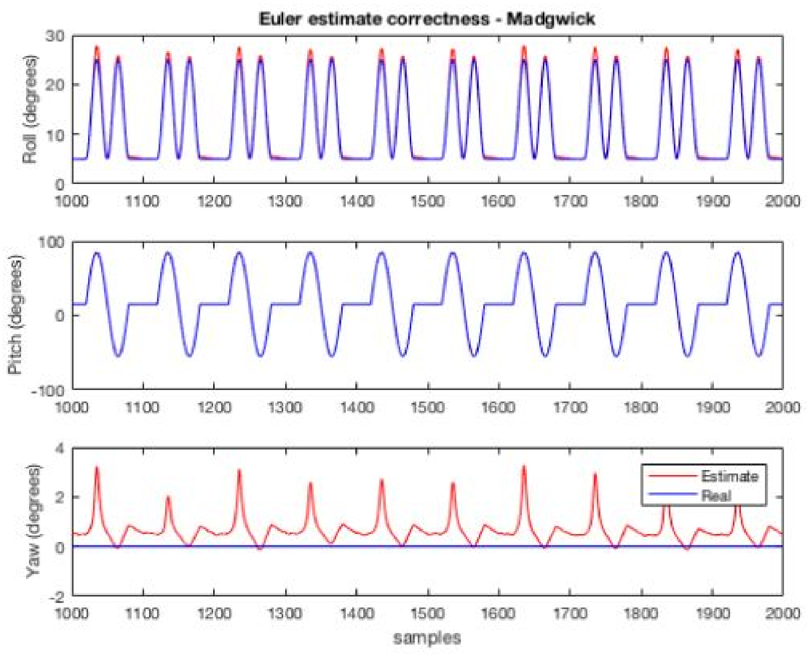


Figure 4.7: Madgwick filter orientation accuracy, pedestrian implementation [6]

For this implementation, a Madgwick filter was chosen. It provides a good orientation estimate and has a low computational cost, only has one filter gain to tune, is specifically optimized to work with IMUs, and the original Madgwick paper contains a C code implementation of the filter [5]. More specifics on this filter and its implementation can be found in Section 4.4.

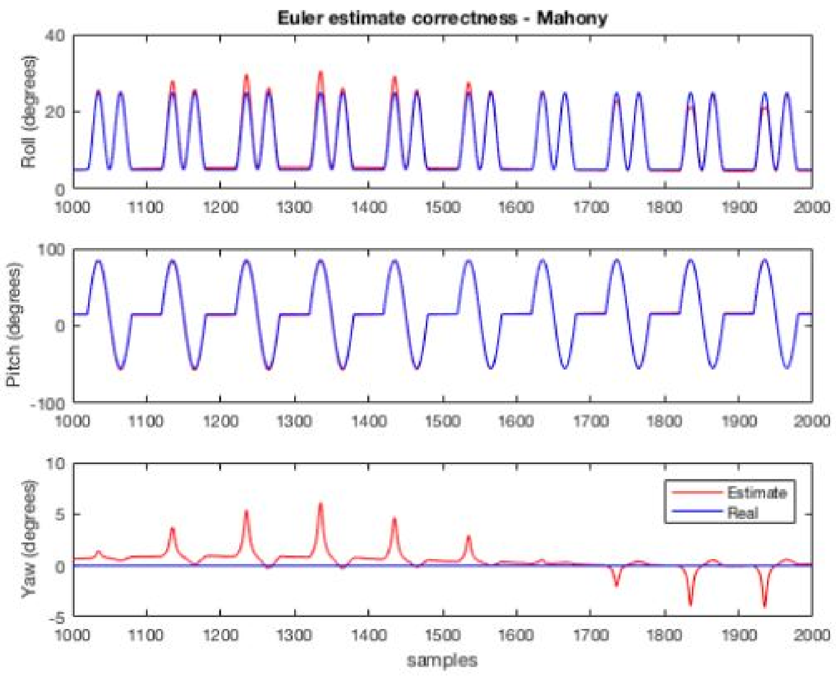


Figure 4.8: Mahony filter orientation accuracy, pedestrian implementation [6]

4.2 IMU Selection and Interfacing

The selection of the MEMS IMU was critical to obtaining an accurate orientation estimate. A MEMS IMU was chosen over other orientation sensors due to its low cost, easy interfacing, and widespread use throughout hobbyist and commercial applications (see Section 4.1.1). However, throughout testing, the IMU was by far the most inaccurate component; for a full underwater implementation, a more reliable and accurate IMU (either high-end MEMS or low-end fiber optic gyroscope) would greatly improve positional accuracy.

There are many MEMS IMUs available for consumers and hobbyists; the most common are the MPU-6050 and the MPU-9050 [10]. These IMUs have open-source libraries written in multiple embedded programming languages and are featured in many hobby projects, but tend to be fairly inaccurate compared to other offerings; additionally, many clones of the sensor exist and it is difficult to identify genuine sensors when buying from third-party retailers. For example, out of six MPU-6050s purchased for the Fo-SHIP in Chapter **??**, only one gave accurate accelerometer information. Other IMUs like the Bosch BNO055 are generally more reliable and provide more accurate measurements; the BNO055 even has a sensor fusion algorithm built in to the chip [4]. While a good step up from the MPU-9050, its sensor fusion algorithm is closed-source, and couldn't be modified for a particular use case.

After much research, a two-part IMU was chosen for its low cost, high accuracy, and availability of pre-written C drivers: the STMicroelectronics LSM6DSOX 3-axis accelerometer and gyroscope and LIS3MDL 3-axis magnetometer. Both ICs are available on a single development board from Adafruit (an online electronics retailer) with documentation and examples for the development board readily available [11]. Both the LSM6DSOX and LIS3MDL are automotive-rated, have low power consumption, and have relatively high sensitivities and data rates [12] [13]. Both communicate using the I^2C protocol, which makes interfacing with the STM32 microcontroller much easier. For a full underwater implementation, a different communications protocol may be desired; the current I^2C implementation requires a request to be sent from the microcontroller to either sensor before sensor data is sent, which reduces the maximum data transmission rate of all three IMU sensors.

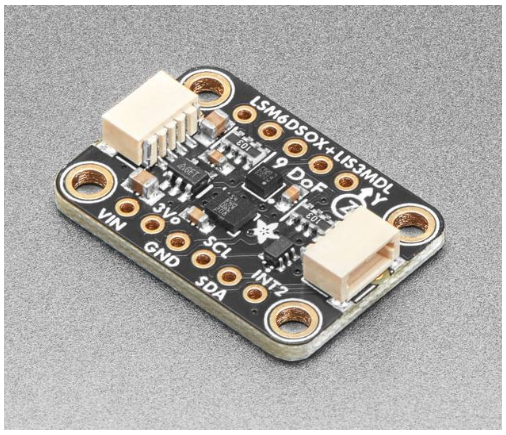


Figure 4.9: LSM6DSOX + LIS3MDL IMU [11]

An important note should be made of the IMU development board's orientation relative to the iSBL array. Figure ?? in Section ?? shows the coordinate frame of the iSBL array, and Figure ?? in Section ?? shows the fully-assembled iSBL array; close inspection shows that the IMU is mounted at a 45° angle relative to the y-axis. This is due to poor planning - the original design had the iSBL array rotated 45° about the x-axis, with the microphone arms forming a "+" (instead of the current "x"). In this configuration, the Fo-SHIP was unable to pitch forward due to the bottom microphone interfering with the lower platforms. So, to get the IMU data in the correct coordinate frame, a transformation must be applied; this transformation is covered in more detail below. Figure 4.10 shows the IMU coordinate frame overlaid on the iSBL array coordinate frame.

Interfacing with the IMU involves communicating with both ICs independently; thankfully, STMicroelectronics provides platform-independent drivers for both sensors, along with example implementations [14] [15]. The interfacing functions described below are modified versions of the provided drivers. Each sensor requires four unique functions to interface using I^2C. Additionally, each sensor has its own I^2C address, even though they are on the same development board.

There are four functions used for interfacing with the LSM6DSOX:
- `lsm6dsox_write()`
- `lsm6dsox_read()`
- `lsm6dsox_read_data_drdy()`
- `lsm6dsox_read_data_drdy_handler()`

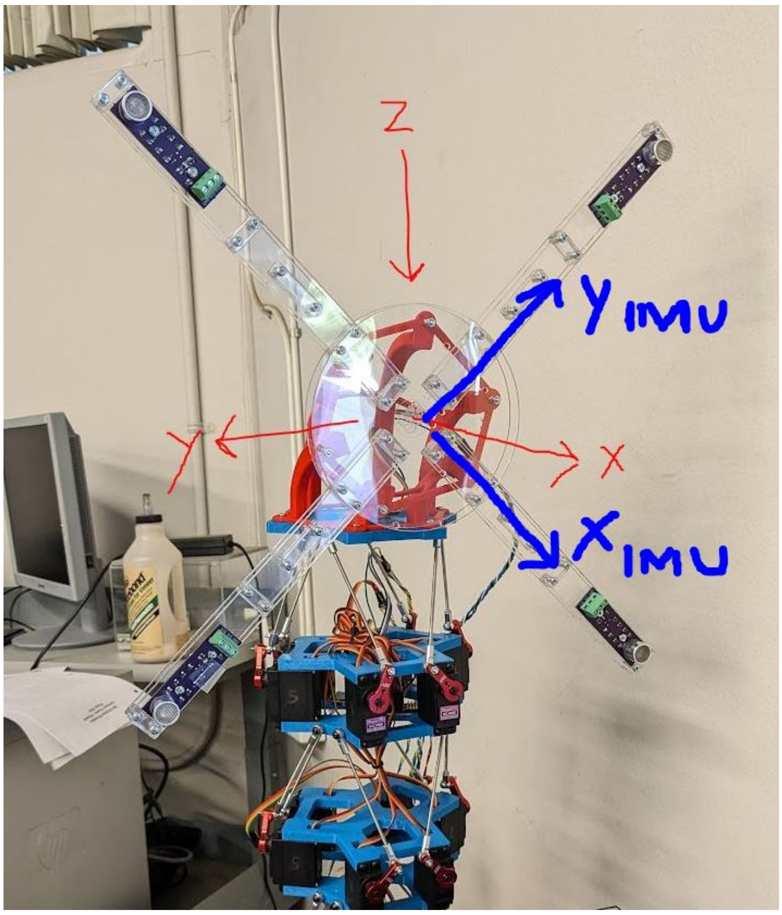


Figure 4.10: Raw IMU coordinate frame (blue) and iSBL array coordinate frame (red)

The first two functions are simple wrappers for the STM32 I^{2}C read/write functions. Because the drivers provided by STMicroelectronics are platform-independent, wrappers are used for the hardware-specific functions. These functions are where the I^{2}C address of the LSM6DSOX is declared.

The next function, `lsm6dsox_read_data_drdy()`, is the initialization function for the LSM6DSOX. In this function, the data rates, accelerometer and gyroscope scales, and data transmission settings are declared. Both the accelerometer and gyroscope are set to update at 208Hz; anything above this was too fast for the STM32 microcontroller, since it is also running the acoustic position estimation code (and updating the Madgwick filter every time the IMU sends data).

The last function, `lsm6dsox_read_data_drdy_handler()`, reads the current acceleration and angular velocity data from the LSM6DSOX. The acceleration data is converted from mg (milli-g's) to g (1g = 32.174 ft/s^2), and the angular velocity data is converted from mdps (milli-degrees per second) to dps. As stated in a previous paragraph, the coordinate frame of the IMU needs to be converted to the iSBL array coordinate frame. This is achieved using a linear combination for the y- and z-axes of the array frame, and a simple axis swap for the x-axis. The raw accelerometer data is saved in a global list `acceleration_g[]`, and the raw gyroscope data is saved in a global list `angular_rate_dps[]`. Finally, if the gyroscope has been calibrated, calibration biases are subtracted from each element of `angular_rate_dps[]`.

```c
void lsm6dsox_read_data_drdy_handler(void)
{
  uint8_t reg;

  /* Read output only if new xl value is available */
  lsm6dsox_xl_flag_data_ready_get(&dev_ctx, &reg);

  if (reg) {
    /* Read acceleration field data */
    memset(data_raw_acceleration, 0x00, 3 * sizeof(int16_t));
    lsm6dsox_acceleration_raw_get(&dev_ctx,data_raw_acceleration);
    // Note: acceleration data is negative by default
    // Z-axis reading to X
    acceleration_g[0] = -lsm6dsox_from_fs4_to_mg(
   data_raw_acceleration[2]) / 1000.0;

    // Combination of X and Y to Y
    acceleration_g[1] = 0.7071 * (lsm6dsox_from_fs4_to_mg(
   data_raw_acceleration[0]) / 1000.0 + lsm6dsox_from_fs4_to_mg(
   data_raw_acceleration[1]) / 1000.0);

    // Combination of X and Y to Z
    acceleration_g[2] = 0.7071 * (-lsm6dsox_from_fs4_to_mg(
   data_raw_acceleration[0]) / 1000.0 + lsm6dsox_from_fs4_to_mg(
   data_raw_acceleration[1]) / 1000.0);
  }

  lsm6dsox_gy_flag_data_ready_get(&dev_ctx, &reg);

  if (reg) {
    /* Read angular rate field data */
    memset(data_raw_angular_rate, 0x00, 3 * sizeof(int16_t));
    lsm6dsox_angular_rate_raw_get(&dev_ctx,data_raw_angular_rate);

    // Z-axis reading to X
    angular_rate_dps[0] = lsm6dsox_from_fs1000_to_mdps(
   data_raw_angular_rate[2]) / 1000.0;

    // Combination of X and Y to Y
    angular_rate_dps[1] = 0.7071 * (-lsm6dsox_from_fs1000_to_mdps(
   data_raw_angular_rate[0]) / 1000.0 -
   lsm6dsox_from_fs1000_to_mdps(data_raw_angular_rate[1]) /
   1000.0);

    // Combination of X and Y to Z
    angular_rate_dps[2] = 0.7071 * (lsm6dsox_from_fs1000_to_mdps(
   data_raw_angular_rate[0]) / 1000.0 -
   lsm6dsox_from_fs1000_to_mdps(data_raw_angular_rate[1]) /
   1000.0);

    // Apply gyroscope calibration only if it has already been
   calibrated
    if (imu_calibrated){
      angular_rate_dps[0] += g_cal[0];
      angular_rate_dps[1] += g_cal[1];
      angular_rate_dps[2] += g_cal[2];
    }
  }
}
```

The LIS3MDL has four functions to get magnetic field strength readings:

- `lis3mdl_write()`
- `lis3mdl_read()`
- `lis3mdl_read_data_drdy()`
- `lis3mdl_read_data_drdy_handler()`

The first two functions, `lis3mdl_write()` and `lis3mdl_read()`, are nearly identical to their LSM6DSOX counterparts; the only difference is the I^2C address.

The third function, `lis3mdl_read_data_drdy()`, is the initialization function for the LIS3MDL. It is similar to its LSM6DSOX counterpart, but uses a different sampling rate of 80Hz - the maximum sampling rate for the magnetometer.

The final function, `lis3mdl_read_data_drdy_handler()`, reads the magnetic field strength data from the LIS3MDL. Both the LIS3MDL and LSM6DSOX share the same coordinate frame, so the transform to the iSBL array coordinate frame is the same as in the `lsm6dsox_read_data_drdy_handler()` function.

When the LSM6DSOX updates every 4.81ms, the INT2 pin on the development board is pulsed. This pin is connected to a GPIO pin on the STM32 which is set up in interrupt mode. When the interrupt is triggered (and if the IMU has been set up), the STM32 stops whatever task it is running and executes the following:

```c
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (imu_rdy){
    if (GPIO_Pin == GPIO_PIN_8)
    {
      lis3mdl_read_data_drdy_handler();
      lsm6dsox_read_data_drdy_handler();
      data_rdy = 1;
      if (imu_calibrated && !recording){
        update_filter();
      }
    }
  }
}
```

Note that since the accelerometer and gyroscope update rate is more than double the magnetometer update rate, the same magnetometer measurements get used multiple times. This is a necessary simplification, as the current Madgwick filter implementation requires all nine measurements (3-axis for each sensor) to perform an update. If the system were switched to a tightly-coupled extended Kalman filter as mentioned in Section 4.1.2, it would be possible (and preferred) to update the measurements separately.

4.3 IMU Setup, Calibration, and Initialization

The IMU is set up and calibrated in the initialization state of the STM32 program, and an initial orientation is formed after 10,000 measurements. The set-up and calibration are performed by one function, `init_imu()`.

```c
void init_imu(void){
  lis3mdl_read_data_drdy();
  lsm6dsox_read_data_drdy();
  lis3mdl_read_data_drdy_handler();
  lsm6dsox_read_data_drdy_handler();
  gyro_calibration(g_cal, 256);
}
```

First, the initialization functions for the LIS3MDL and LSM6DSOX are called. At the end of the `lsm6dsox_read_data_drdy()` function, a global flag `imu_rdy` is set to `1`. Before this step, the IMU is not configured for the proper data rates or sensitivities, so it should not be read from. Then, the data reading functions for each IC are called once; this triggers both sensors to start updating their measurements at their set data rates.

Calibrating a MEMS IMU is one of the most important steps in getting accurate data [3]. The three sensors on the IMU have known sources of error that can be mitigated: a gyroscope at rest should not measure any angular velocity; an accelerometer at rest should only measure gravity [16]; and a magnetometer rotated about all its axes should measure an equal, non-skewed magnetic field [17].

To calibrate the gyroscope, the IMU is held at rest and multiple (256, in this implementation) measurements are taken. Assuming that the IMU is truly at rest, the readings for each axis should average to zero; noise from the environment and sensor will produce readings that are not perfectly zero, but taking an average of a large number of samples mitigates this noise. However, MEMS gyroscopes tend to have some constant offset (called a bias) when at rest. This bias can be measured and subtracted from subsequent readings after calibration is complete. Note that the calibration function is a blocking function in this implementation and relies on the LSM6DSOX triggering interrupts on the INT2 pin.

```c
void gyro_calibration(float g_cal[3], int num_iter){
  for (int i = 0; i < num_iter; i++){
    while (!data_rdy);
    g_cal[0] -= angular_rate_dps[0];
    g_cal[1] -= angular_rate_dps[1];
    g_cal[2] -= angular_rate_dps[2];
    data_rdy = 0;
  }
  g_cal[0] /= (float)num_iter;
  g_cal[1] /= (float)num_iter;
  g_cal[2] /= (float)num_iter;
  imu_calibrated = 1;
}
```

Many of the calibration models for IMUs assume a linear model for measurements, like that in Equation 4.2. This assumes that the true value of the measurement can be described by the raw measurement `x` times a gain `K` plus a bias `b`. Often, these are formulated with measurements and bias as vectors and gains as matrices [16].

$$\mathbf{x}_{calibrated} = \mathbf{K}\mathbf{x}_{raw} + \mathbf{b} \tag{4.2}$$

In a complete system, each sensor should be calibrated according to this model. For this implementation, only the gyroscope is calibrated and only a bias term is considered. Measuring the gain matrix for the gyroscope is not possible without external equipment.

Calibrating the accelerometer and magnetometer requires rotating the IMU in a controlled manner. The Fo-SHIP may appear capable of doing this, but calibration routines require placing the IMU level on its six faces (for the accelerometer) and rotating it around in every possible orientation (for the magnetometer). This motion is not possible with the Fo-SHIP, and the IMU has been permanently affixed to the iSBL array with adhesive prior to calibration being considered. With better planning, it would have been possible to calibrate the IMU before attaching it to the array. This approach is highly recommended for future implementations, and calibration routines for the accelerometer [16] and magnetometer [17] are available in the references of this thesis.

The final step in the initialization of the IMU is measuring an initial orientation estimate. As mentioned in Section **??**, the yaw component of the initial orientation must be removed to transform the acoustic position estimates from the global frame to the test frame. Before the first Madgwick filter update, the orientation estimate is simply the unit quaternion $\mathbf{q}_0 = [1, 0, 0, 0]$. This estimate is slowly affected by new IMU measurements; it doesn't automatically move to the true orientation of the platform. After a set number of filter updates (10,000, in this implementation), the Madgwick filter will have converged on a good orientation estimate - the quaternion is then saved and the initial yaw is extracted. The number of iterations required depends on the filter gain `beta`, the update frequency of the IMU, and the initial orientation relative to the global frame.

```c
void update_filter(void)
  ...
  // if the filter has converged, set the initial quaternion
  if (imu_i == -1){
    initQuat.w = q0;
    initQuat.x = -q1;
    initQuat.y = -q2;
    initQuat.z = -q3;
    init_yaw = extract_yaw(initQuat);
    imu_init_set = 1;
    imu_i = 0;
  }
}
```

It would be highly recommended to modify the Madgwick filter to speed up this initial orientation estimation process. For example, the initial quaternion (before the first filter update) could be set based on an average of the first five accelerometer and magnetometer readings. This was not implemented due to time constraints (and because the current approach works well enough).

4.4 Madgwick Filter

This section details the Madgwick filter, the sensor fusion algorithm used to combine accelerometer, gyroscope, and magnetometer measurements into an absolute orientation estimate. It is split into three subsections: the first discusses quaternions, a mathematical tool used to describe the orientation of a coordinate frame in three-dimensional space; the second describes the theory of the Madgwick filter; and the final shows the code implementation of the filter. This implementation draws on Sebastian Madgwick's original paper, "An efficient orientation filter for inertial and inertial/magnetic sensor arrays" [5].

4.4.1 Quaternion Representation

Quaternions are a mathematical concept with many applications; for this thesis, only the subset called "rotation quaternions" will be considered. Any reference to a "quaternion" for the remainder of this paper will be referring to rotation quaternions [18].

Euler's rotation formula states that any 3D rotation can be described with two parameters: a unit vector defining the axis of rotation, and an angle describing the magnitude of rotation about that axis [18]; see Figure 4.11 for a graphical representation. Quaternions are related to the angle-axis representation by the following set of equations [18]:

$$\mathbf{q} = [q_0, q_1, q_2, q_3] \tag{4.3}$$

$$q_0 = \cos\left(\frac{\theta}{2}\right) \tag{4.4}$$

$$q_1 = \hat{x}\sin\left(\frac{\theta}{2}\right) \tag{4.5}$$

$$q_2 = \hat{y}\sin\left(\frac{\theta}{2}\right) \tag{4.6}$$

$$q_3 = \hat{z}\sin\left(\frac{\theta}{2}\right) \tag{4.7}$$

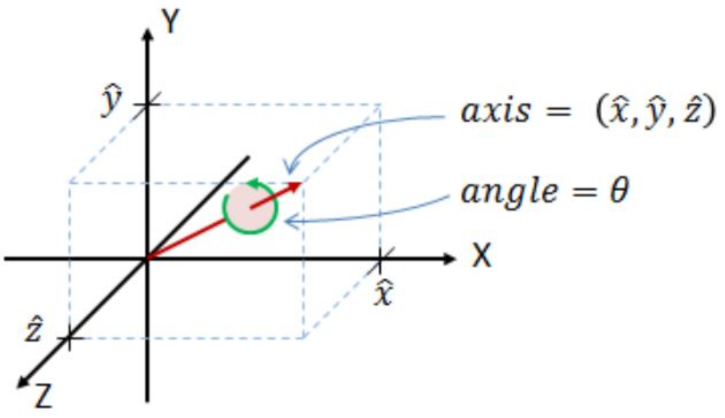


Figure 4.11: Angle-axis representation of 3D rotation [18]

Two other concepts are often used to represent the rotation between two coordinate frames: rotation matrices and Euler angles (pitch, roll, yaw). Rotation matrices are useful for rotating 3D points and are discussed in Sections **??** and **??**. Euler angles are the most common way of describing rotations (particularly in aircraft) due to their simplicity and ease-of-understanding. However, Euler angles depend on a particular sequence of rotations to take place (generally, yaw-pitch-roll, in that order) and are subject to a phenomenon known as "gimbal lock." In gimbal lock, when the pitch angle of a body reaches ±90°, a singularity occurs in the equations for the Euler angles; the yaw and roll axes become indistinguishable from each other [18]. To avoid dealing with this special case, the Madgwick filter uses quaternions to describe rotations between bodies [5].

Specific uses of quaternions, including rotating a point by a quaternion, rotating a quaternion by another quaternion, and converting between the Euler angle representation and quaternion representation of a rotation can be found in Section **??**. For the remainder of this section, quaternions will be used to describe any rotations between coordinate frames.

4.4.2 Filter Theory

Using an accelerometer, gyroscope, and magnetometer, two different orientation estimates can be formed: one from the relative change in orientation described by gyroscope readings, and one from the absolute measurements of the accelerometer (gives “down” direction) and the magnetometer (gives “magnetic north” direction). For each of these two absolute measurements, two vectors can be defined: one that points towards the true position of the quantity being measured in the global frame (gravity for accelerometer, and magnetic north for magnetometer), and one that points in the direction of the sensor measurement. For example, if the IMU was at a pitch angle of 30° about the y-axis, the vector in Equation 4.8 shows true gravity in the global frame and the vector in Equation 4.9 shows what the accelerometer would measure (assuming a NED coordinate frame) [5].

$$\mathbf{d} = [0, 0, g] \tag{4.8}$$

$$\mathbf{s} = [-g\cos(30), 0, g\sin(30)] \tag{4.9}$$

A minimization problem can be formed for each absolute measurement, where the true vector in the global frame rotated by the current quaternion (orientation estimate of the sensor) should equal the measured vector in the sensor frame. This is represented as an objective function for a single sensor in Equation 4.10, and for the combination of accelerometer and magnetometer in Equation 4.11 [5].

$$\mathbf{f}({}^S_E\hat{\mathbf{q}}, {}^E\hat{\mathbf{d}}, {}^S\hat{\mathbf{s}}) = {}^S_E\hat{\mathbf{q}}^* \otimes {}^E\hat{\mathbf{d}} \otimes {}^S_E\hat{\mathbf{q}} - {}^S\hat{\mathbf{s}} \tag{4.10}$$

$$\mathbf{f}_{g,b}({}^S_E\hat{\mathbf{q}}, {}^S\hat{\mathbf{a}}, {}^E\hat{\mathbf{b}}, {}^S\hat{\mathbf{m}}) = \begin{bmatrix} \mathbf{f}_g({}^S_E\hat{\mathbf{q}}, {}^S\hat{\mathbf{a}}) \\ \mathbf{f}_b({}^S_E\hat{\mathbf{q}}, {}^E\hat{\mathbf{b}}, {}^S\hat{\mathbf{m}}) \end{bmatrix} \tag{4.11}$$

The gradient of the objective function can also be computed; see Equation 4.12 for the general form and Equation 4.13 for the accelerometer / magnetometer combined form. The gradient is defined by the objective function and its Jacobian; the Jacobian of the combined accelerometer / magnetometer objection function can be seen in Equation 4.14. These are used along with the step size μ to compute the accelerometer / magnetometer orientation estimate for the next filter update (see Equation 4.15) [5].

$$\nabla\mathbf{f}({}^S_E\hat{\mathbf{q}}_k, {}^E\hat{\mathbf{d}}, {}^S\hat{\mathbf{s}}) = \mathbf{J}^T({}^S_E\hat{\mathbf{q}}_k, {}^E\hat{\mathbf{d}})\mathbf{f}({}^S_E\hat{\mathbf{q}}_k, {}^E\hat{\mathbf{d}}, {}^S\hat{\mathbf{s}}) \tag{4.12}$$

$$\nabla\mathbf{f} = \begin{Bmatrix} \mathbf{J}_g^T({}^S_E\hat{\mathbf{q}}_{est,t-1})\mathbf{f}_g({}^S_E\hat{\mathbf{q}}_{est,t-1}, {}^S\hat{\mathbf{a}}_t) \\ \mathbf{J}_{g,b}^T({}^S_E\hat{\mathbf{q}}_{est,t-1}, {}^E\hat{\mathbf{b}})\mathbf{f}_{g,b}({}^S_E\hat{\mathbf{q}}_{est,t-1}, {}^S\hat{\mathbf{a}}, {}^E\hat{\mathbf{b}}, {}^S\hat{\mathbf{m}}) \end{Bmatrix} \tag{4.13}$$

$$\mathbf{J}_{g,b}({}^S_E\hat{\mathbf{q}}, {}^E\hat{\mathbf{b}}) = \begin{bmatrix} \mathbf{J}_g^T({}^S_E\hat{\mathbf{q}}) \\ \mathbf{J}_b^T({}^S_E\hat{\mathbf{q}}, {}^E\hat{\mathbf{b}}) \end{bmatrix} \tag{4.14}$$

$$ {}^S_E\mathbf{q}_{\nabla,t} = {}^S_E\hat{\mathbf{q}}_{est,t-1} - \mu_t \frac{\nabla \mathbf{f}}{||\nabla \mathbf{f}||} \tag{4.15} $$

The orientation estimate computed from the gyroscope measurements is described in Equations 4.16 and 4.17. This orientation estimate is then combined with the accelerometer / magnetometer orientation estimate in Equation 4.18. The weighting factor γ_t is calculated to ensure the optimal fusion of the two orientation estimates; this calculation is skipped here for brevity [5].

$$ {}^S_E\dot{\mathbf{q}}_{\omega,t} = \frac{1}{2} {}^S_E\hat{\mathbf{q}}_{est,t-1} \otimes {}^S\boldsymbol{\omega}_t \tag{4.16} $$

$$ {}^S_E\mathbf{q}_{\omega,t} = {}^S_E\hat{\mathbf{q}}_{est,t-1} + {}^S_E\dot{\mathbf{q}}_{\omega,t}\Delta t \tag{4.17} $$

$$ {}^S_E\mathbf{q}_{est,t} = \gamma_t {}^S_E\mathbf{q}_{\nabla,t} + (1-\gamma_t) {}^S_E\mathbf{q}_{\omega,t}, \quad 0 \leq \gamma_t \leq 1 \tag{4.18} $$

In this algorithm, two errors can be compensated for: magnetic distortion and gyroscope bias drift. Hard iron sources, like permanent magnets in electric motors, can distort the magnetic field around the magnetometer and can be removed with proper calibration (as mentioned in Section 4.3). Inclination errors, which are the vertical component of soft iron distortions (caused by stray magnetic fields, such as a field generated by running current through a wire), can be compensated for with the Madgwick filter. This is achieved by rotating the magnetometer measurements by the current orientation estimate and using the components of the resulting vector to form the true magnetic north vector. This process is able to remove the vertical effect of soft-iron distortions, but horizontal effects still remain. Equations 4.19 and 4.20 show this process [5].

$$ {}^E\hat{\mathbf{h}}_t = [0 \quad h_x \quad h_y \quad h_z] = {}^S_E\mathbf{q}_{est,t-1} \otimes {}^S\hat{\mathbf{m}}_t \otimes {}^S_E\mathbf{q}^*_{est,t-1} \tag{4.19} $$

$$ {}^E\hat{\mathbf{b}}_t = \left[0 \quad \sqrt{h_x^2 + h_y^2} \quad 0 \quad h_z\right] \tag{4.20} $$

The gyroscope bias drifts over time due to temperature changes and motion. This bias is initially eliminated by calibrating the gyroscope (as mentioned in Section 4.3), but the Madgwick filter includes a method for removing the drift over time. The drift is estimated using an integral feedback approach: the change in orientation is estimated using the difference between the current and previous orientation estimate, and this estimate is combined with the gyroscope measurements to eliminate the drift over time. Equations 4.21, 4.22, and 4.23 show this process [5].

$$ {}^S\boldsymbol{\omega}_{\epsilon,t} = 2 {}^S_E\hat{\mathbf{q}}^*_{est,t-1} \otimes {}^S_E\dot{\hat{\mathbf{q}}}_{\epsilon,t} \tag{4.21} $$

$$ {}^S\boldsymbol{\omega}_{b,t} = \zeta \sum_t {}^S\boldsymbol{\omega}_{\epsilon,t}\Delta t \tag{4.22} $$

$$^{S}\boldsymbol{\omega}_{c,t} = {}^{S}\boldsymbol{\omega}_{t} - {}^{S}\boldsymbol{\omega}_{b,t} \tag{4.23}$$

Finally, all of the above concepts are combined to form the Madgwick filter. Figure 4.12 shows the block diagram for the full system. Some simplifications not covered in this explanation (particularly for the weighting of the two orientation estimates) are implemented into the diagram. Here, two tunable gains are presented: β, which is used to modify the weighting factor between the two orientation estimates, and ζ, which is used to weight the gyroscope bias drift compensation [5].

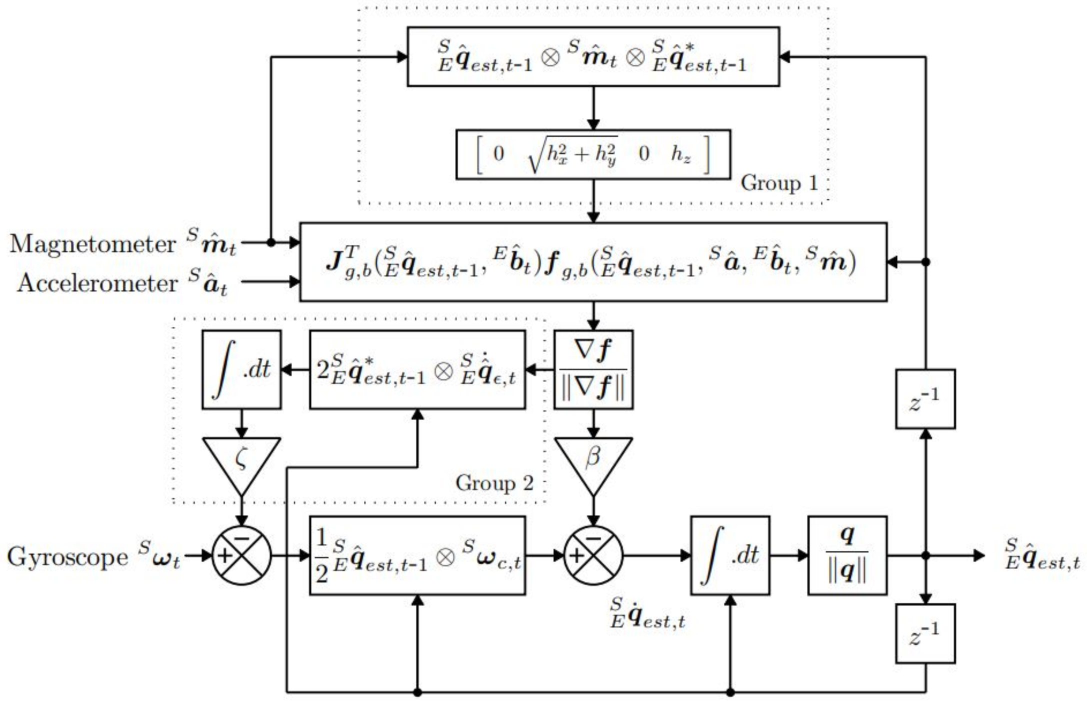

$^S_E\hat{\boldsymbol{q}}_{est,t\text{-}1} \otimes {}^S\hat{\boldsymbol{m}}_t \otimes {}^S_E\hat{\boldsymbol{q}}^*_{est,t\text{-}1}$
$\begin{bmatrix} 0 & \sqrt{h_x^2+h_y^2} & 0 & h_z \end{bmatrix}$
Group 1
Magnetometer $^S\hat{\boldsymbol{m}}_t$
Accelerometer $^S\hat{\boldsymbol{a}}_t$
$\boldsymbol{J}^T_{g,b}({}^S_E\hat{\boldsymbol{q}}_{est,t\text{-}1}, {}^E\hat{\boldsymbol{b}}_t)\boldsymbol{f}_{g,b}({}^S_E\hat{\boldsymbol{q}}_{est,t\text{-}1}, {}^S\hat{\boldsymbol{a}}, {}^E\hat{\boldsymbol{b}}_t, {}^S\hat{\boldsymbol{m}})$
$\int .dt$
$2{}^S_E\hat{\boldsymbol{q}}^*_{est,t\text{-}1} \otimes {}^S_E\dot{\hat{\boldsymbol{q}}}_{\epsilon,t}$
$\frac{\nabla \boldsymbol{f}}{\|\nabla \boldsymbol{f}\|}$
z^{-1}
ζ
β
Group 2
Gyroscope $^S\boldsymbol{\omega}_t$
$\frac{1}{2}{}^S_E\hat{\boldsymbol{q}}_{est,t\text{-}1} \otimes {}^S\boldsymbol{\omega}_{c,t}$
$\int .dt$
$\frac{\boldsymbol{q}}{\|\boldsymbol{q}\|}$
$^S_E\hat{\boldsymbol{q}}_{est,t}$
$^S_E\dot{\boldsymbol{q}}_{est,t}$
z^{-1}

Figure 4.12: Block diagram for Madgwick filter with accelerometer, gyroscope, and magnetometer (error compensation for magnetic distortion in Group 1 and for gyroscope bias drift in Group 2) [5]

4.4.3 Filter Implementation

The code implementation of the Madgwick filter used in this thesis is a modified version of Sebastian Madgwick's C implementation, available on his company's website [19]. This implementation appears to not incorporate the gyroscope bias drift compensation, and there is no corresponding ζ gain used. The code is highly optimized and uses as few scalar arithmetic operations as possible, so some lines may seem fairly complex.

First, the time since the last filter update is calculated using timer TIM13. The timer has a prescalar of 4096 and a period of 65536; it ticks at approximately 67.1kHz and has an overall period of approximately 0.976s. The filter updates based on the update rate of the LSM6DSOX, which is set at 208Hz in this implementation, so there is little worry of exceeding the overall period.

```c
timestamp = __HAL_TIM_GET_COUNTER(&htim13);
uint32_t diff_ticks;
if (timestamp >= previousTimestamp) {
  diff_ticks = timestamp - previousTimestamp;
} else {
  diff_ticks = (65536 - previousTimestamp) + timestamp;
}
deltat = (float)diff_ticks * 4096.0f / 275000.0f;
previousTimestamp = __HAL_TIM_GET_COUNTER(&htim13);
```

The data from the IMU is then saved in nine floats. The gyroscope readings are converted from degrees per second to radians per second, as required by the filter. The units of the accelerometer and magnetometer do not matter since they are normalized, but they are saved in g's and Gauss, respectively.

```c
float ax = acceleration_g[0];
float ay = acceleration_g[1];
float az = acceleration_g[2];
float gx = angular_rate_dps[0] * 3.14159 / 180.0;
float gy = angular_rate_dps[1] * 3.14159 / 180.0;
float gz = angular_rate_dps[2] * 3.14159 / 180.0;
float mx = magnetic_G[0];
float my = magnetic_G[1];
float mz = magnetic_G[2];
```

The rate of change of the quaternion is calculated from the gyroscope data as described in Equation 4.16 [19].

```c
qDot1 = 0.5f * (-q1 * gx - q2 * gy - q3 * gz);
qDot2 = 0.5f * (q0 * gx + q2 * gz - q3 * gy);
qDot3 = 0.5f * (q0 * gy - q1 * gz + q3 * gx);
qDot4 = 0.5f * (q0 * gz + q1 * gy - q2 * gx);
```

After this, the accelerometer and magnetometer measurements are normalized. Some auxiliary variables are computed to avoid repeating arithmetic (such as the product of q0 and q1, for example).

Next, the magnetic distortion calibration described in Equations 4.19 and 4.20 is implemented [19].

```c
hx = mx * q0q0 - _2q0my * q3 + _2q0mz * q2 + mx * q1q1 + _2q1 * my
     * q2 + _2q1 * mz * q3 - mx * q2q2 - mx * q3q3;
hy = _2q0mx * q3 + my * q0q0 - _2q0mz * q1 + _2q1mx * q2 - my *
    q1q1 + my * q2q2 + _2q2 * mz * q3 - my * q3q3;
```

The gradient descent algorithm steps in Equations 4.10 through 4.15 are performed in a highly-optimized manner. These return the computed gradient of the accelerometer / magnetometer quaternion, $\nabla \mathbf{f}$ [19].

```c
s0 = -_2q2 * (2.0f * q1q3 - _2q0q2 - ax) + _2q1 * ...
s1 = _2q3 * (2.0f * q1q3 - _2q0q2 - ax) + _2q0 * ...
s2 = -_2q0 * (2.0f * q1q3 - _2q0q2 - ax) + _2q3 * ...
s3 = _2q1 * (2.0f * q1q3 - _2q0q2 - ax) + _2q2 * ...
```

The gradient is normalized to give the estimated rate of change of the quaternion. Next, the accelerometer / magnetometer estimate is combined with the estimated rate of change of the quaternion from the gyroscope readings, using β as a weighting factor [19].

```
qDot1 -= beta * s0;
qDot2 -= beta * s1;
qDot3 -= beta * s2;
qDot4 -= beta * s3;
```

Finally, this combined rate of change estimate is integrated using the `deltat` term computed earlier, and is added to the previous orientation estimate quaternion [19].

```
q0 += qDot1 * deltat / 1000;
q1 += qDot2 * deltat / 1000;
q2 += qDot3 * deltat / 1000;
q3 += qDot4 * deltat / 1000;
```

The updated quaternion is then normalized and can be used as the current orientation estimate of the sensor / iSBL array. The filter runs in the background as the STM32 runs the acoustic positioning system code and is paused when recording from the ADC.

4.5 Dead-Reckoning

Dead reckoning is a ubiquitous and extremely important navigation technique. Before the creation of GPS satellites, it was a primary method for navigating the high seas when celestial navigation was not an option. Similarly, airplane pilots and navigators would use dead reckoning to travel from place to place, taking into account the plane's relative speed and the velocity of the air around them [20]. Hikers and explorers make use of dead reckoning when they use a compass to determine their heading and then walk a certain distance in that direction. Dead reckoning is simply the technique of using time, heading, velocity, and external factors to determine the change in position from a known initial position.

Though GPS and GNSS have drastically changed how the world is navigated, dead reckoning is still commonly used. Submarines that only use GPS for absolute location fixes (as opposed to acoustic positioning methods or similar) often use doppler velocity logs, which measure the velocity of the submarine relative to the ocean floor; when combined with gyromagnetic compass headings, dead reckoning can be used produce a position estimate [21]. Pedestrian movements can be estimated using the IMUs in modern smartphones, which allows for position estimation when external sources (WiFi positioning, GPS) are unavailable [6] [22].

Dead reckoning is an inherently error-prone technique: errors in the initial position estimate, speed (or acceleration) measurements, and heading measurements all propagate into future position estimates. To minimize these errors, extremely accurate and calibrated measurement equipment is recommended; for IMUs, Figure 4.3 in Section 4.1.1 shows how the performance grade of an IMU correlates to its dead reckoning accuracy. The IMU chosen for this thesis falls between the consumer and industrial performance grades; for this grade, dead reckoning is very inaccurate [3]. Despite this limitation, an attempt at dead reckoning using the IMU will be made.

Using IMUs for dead reckoning is inherently more error-prone due to an additional integration requirement. Most dead reckoning methods on ships, submarines, and aircraft use heading and velocity; IMUs measure heading and acceleration. With velocity, only one integration is required to give position, but acceleration requires two. This means that any minuscule error in accelerometer measurements will be propagated into a large position error.

To combat this, a few assumptions are made about the movement of the Fo-SHIP. First, the movement is assumed to be relatively quick - the Fo-SHIP moves from one position to the next in under one second, and then holds that position until another movement is executed. Second, the motion of the Fo-SHIP is not very smooth; the Fo-SHIP interpolates between the initial and final position in discrete steps, and due to the serial nature of the hexapod platforms, any jitter in lower platforms is propagated up to the end effector. As a result, the accelerometer is rarely at rest during movement between set points. Finally, it is assumed that the Fo-SHIP starts at rest and ends at rest; any integrated velocity should start at zero and end at zero, and any non-zero velocity at the end of a move should be considered velocity drift.

These assumptions are quite similar to those made in pedestrian dead reckoning, often known as "gait tracking." When walking, a pedestrian's foot starts at rest, moves with large accelerations, and ends at rest. Gait tracking does make use of other assumptions (relatively standardized step sizes among pedestrians, known orientation of the pedestrians' feet, etc.), but it provides a good starting point for implementing dead reckoning on the Fo-SHIP. For a full underwater implementation, these assumptions would not hold; motion in the water is much more smooth and the underwater vehicle is almost never completely stationary. However, underwater implementations can make use of additional sources of information (doppler velocity logs, input to thrusters, speed of the vehicle relative to the water around it) which are not available in the above-water implementation. For the underwater implementation, making use of these information sources and removing the gait tracking assumptions would be highly encouraged.

The approach taken in this thesis builds off of another Sebastian Madgwick project, "Gait Tracking with x-IMU" [22]. This project is written in Matlab and includes some assumptions not valid for the thesis implementation, so the code has been re-worked significantly. The overall goals and workflow of this implementation are as follows:

- Rotate the accelerometer measurements from the sensor frame to the test frame
- Determine if the array is accelerating
- Integrate acceleration measurements into a change in velocity
- Compensate for velocity drift if the array is stationary
- Integrate the current velocity estimate into a change in position

First, the accelerometer measurements are rotated from the sensor frame to the test frame. The dead reckoning code runs immediately after the Madgwick filter updates, so the orientation of the sensor frame relative to the test frame is already available. The rotation of the measurements follows the same procedure described in Section **??**: rotate the current orientation estimate by the yaw component of the initial orientation to transform from the global frame to the test frame; normalize the quaternion; and then rotate the acceleration measurement vector by the normalized quaternion.

```c
// extract the current orientation of the platform
Quaternion quat_raw = {q0, q1, q2, q3};

// create a quaternion to undo initial yaw rotation and apply it
    to the current quaternion
Quaternion yaw_compensation = create_yaw_quaternion(init_yaw);
Quaternion quat = multiply_quaternions(yaw_compensation,quat_raw);

// normalize the result to ensure it's a valid rotation
normalize_quaternion(&quat);

// rotate the accelerometer readings by the current quaternion to
    get the acceleration from the sensor frame to the global frame
Point3D acc_temp = {acceleration_g[0], acceleration_g[1],
    acceleration_g[2]};
Point3D acc_corr_temp = rotatePoint(acc_temp, quat);
```

The next step is to determine if the platform is accelerating. As stated above, it is assumed that the platform is moving if the accelerometer readings are significant enough. The absolute value magnitude of the accelerometer readings is calculated and gravity is subtracted from the magnitude to ensure only non-gravitational acceleration is considered.

```c
float acc_mag_abs = fabsf(sqrtf(acceleration_g[0]*acceleration_g
    [0] + acceleration_g[1]*acceleration_g[1] + acceleration_g[2]*
    acceleration_g[2]) - 1);
```

Then, the acceleration magnitude is compared to an experimentally-determined threshold value. MEMS IMUs tend to have fairly noisy measurements, and a stationary sensor may produce a non-zero acceleration magnitude; this threshold ensures that only significant movements are considered. If the acceleration magnitude exceeds the threshold, then the rotated acceleration measurements are integrated into a change in velocity. The `deltat` value is the time since the last Madgwick filter update in milliseconds, as described in Section 4.4.3. This thresholding removes considerable drift from the dead reckoning position estimate.

```c
if (acc_mag_abs > 0.04){
  // acceleration is in g's, convert to m/s^2
  v_corrected.x -= (acc_corr_temp.x) * deltat/1000 * 9.81;
  v_corrected.y -= (acc_corr_temp.y) * deltat/1000 * 9.81;
  v_corrected.z -= (acc_corr_temp.z - 1) * deltat/1000 * 9.81;
  acc_stable_cnt = 0;
  acc_stable_flag = 1;
}
```

If the threshold is exceeded, then the IMU is considered “non-stable” - it is currently moving relative to the test frame. If it doesn’t exceed the threshold, then it is considered “stable.” If the IMU has been stable for a set number of filter updates after movement is detected (20, in this implementation), then velocity drift compensation is applied. As stated above, another assumption is that if the IMU has been stable for a period of time, it is considered to be stationary; in this case, the velocity should equal zero. This is rarely the case due to velocity drift.

For ease of implementation, the velocity drift is assumed to be some constant bias that adds to each velocity measurement. This bias results in the accumulated velocity ending at some non-zero value when the IMU becomes stable after moving. Figure 4.13 provides a visualization of this phenomenon, with the line representing `v_corrected[]` over time.

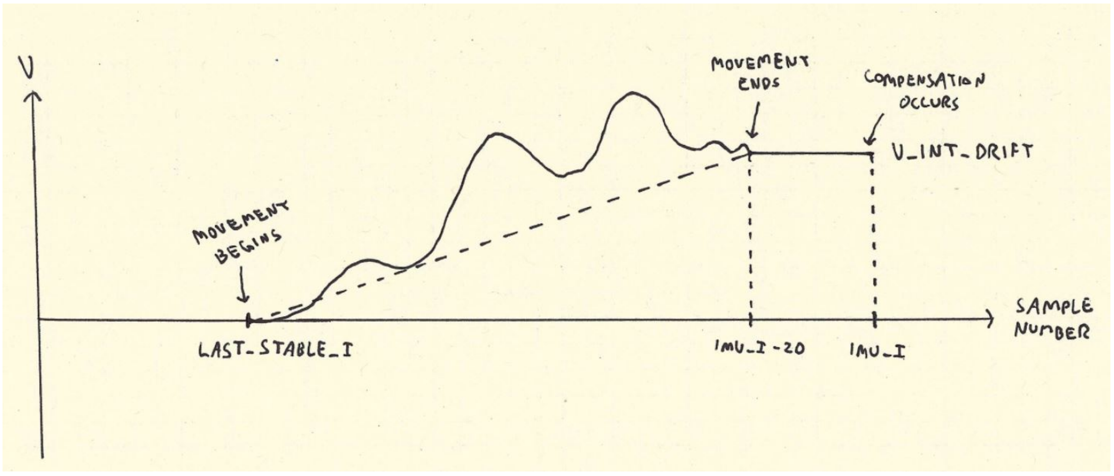


Figure 4.13: IMU velocity drift example, 1D case

The change in position can be described as the area underneath this velocity curve (or the integration over time). This constant velocity drift can be removed from the change in position by subtracting the area of two regions on the graph: the triangular region during the time of motion, and the rectangular region when the IMU is stationary. This is only possible because the current velocity of the IMU is integrated into a change in position at every time step, as seen at the end of this section.

First, the value of `v_int_drift[]` is calculated as the difference between the initial velocity and the final velocity. For this implementation, the initial velocity will always be zero - however, a vector `v_corrected_last_stable[]` is used to represent the initial velocity in case this implementation is modified.

```
Point3D v_int_drift;
v_int_drift.x = v_corrected.x - v_corrected_last_stable.x;
v_int_drift.y = v_corrected.y - v_corrected_last_stable.y;
v_int_drift.z = v_corrected.z - v_corrected_last_stable.z;
```

Next, the time between the start and end of motion is calculated. `imu_i` is a counter that tracks how many filter updates have occurred (and overflows at 10,000 iterations), and `last_stable_i` saves the index of when the IMU started to move. `imu_diff` is the number of filter updates between the start of movement and the end of movement of the IMU (the width of the triangle in Figure 4.13).

```
float i_diff = (imu_i > last_stable_i) ? (float) imu_i -
    last_stable_i : (float) 10000 - last_stable_i + imu_i;
i_diff += 1;
i_diff -= (float) acc_stable_cnt;
```

Next, the area of the two regions is subtracted from the current change-in-position estimate. Since `deltat` can fluctuate, the update rate of the LSM6DSOX (208Hz) is used to convert the sample count to time. This is a simplification, and summing the values of `deltat` for each iteration that the IMU is moving would give a more precise result.

```
delta_x_imu.x -= (v_int_drift.x * ((i_diff / 2) + (float)
    acc_stable_cnt) / sampleFreq);
delta_x_imu.y -= (v_int_drift.y * ((i_diff / 2) + (float)
    acc_stable_cnt) / sampleFreq);
delta_x_imu.z -= (v_int_drift.z * ((i_diff / 2) + (float)
    acc_stable_cnt) / sampleFreq);
```

Finally, the velocity drift value is subtracted from the current velocity estimate. This always resets `v_corrected[]` to the value of the last stable velocity `v_corrected_last_stable[]`, which in practice is always zero. A flag, `acc_stable_flag`, is used to ensure that the velocity drift compensation only occurs once for every chunk of movement; here, it is reset to zero.

```
v_corrected.x -= v_int_drift.x;
v_corrected.y -= v_int_drift.y;
v_corrected.z -= v_int_drift.z;
acc_stable_flag = 0;  // until the system is unstable again, don't
     re-account for velocity drift
```

If the IMU is stable and the velocity drift compensation has already been applied, then the `last_stable_i` is updated to the current `imu_i` value. Additionally, the last stable velocity is set to the current velocity (always zero in this implementation).

```
else if (!acc_stable_flag){
  last_stable_i = imu_i;
  v_corrected_last_stable.x = v_corrected.x;
  v_corrected_last_stable.y = v_corrected.y;
  v_corrected_last_stable.z = v_corrected.z;
}
```

Lastly, for every filter update (does not matter if the IMU is stable or moving), the change in position is updated using the current velocity of the IMU. This is the second integration step that occurs, following the first integration of acceleration to velocity in the beginning of the code.

```
delta_x_imu.x += v_corrected.x * deltat/1000;
delta_x_imu.y += v_corrected.y * deltat/1000;
delta_x_imu.z += v_corrected.z * deltat/1000;
```

The vectors for the change in position, corrected velocity, and last stable velocity, as well as the counters `imu_i` and `last_stable_i`, are reset after a complete acoustic position estimate. The values are all set to zero after the STM32 sends its position estimate data to the ESP32 and Fo-SHIP.

```
v_corrected.x = 0;
v_corrected.y = 0;
v_corrected.z = 0;
delta_x_imu.x = 0;
delta_x_imu.y = 0;
delta_x_imu.z = 0;
v_corrected_last_stable.x = 0;
v_corrected_last_stable.y = 0;
v_corrected_last_stable.z = 0;
imu_i = 0;
last_stable_i = 0;
```

The result of the dead reckoning is the change in position since the last acoustic position estimate, and it is stored in `delta_x_imu[]`. This relative change in position is fed into a Kalman filter along with the absolute acoustic position estimate. Section **??** details how this is accomplished.

4.6 System Validation

After being implemented in code, both the orientation estimation system (the Madgwick filter) and the dead reckoning system were tested and validated to ensure accuracy. The Madgwick filter implementation has one gain, `beta`, that must be tuned. Higher values of `beta` imply more trust in the accelerometer and magnetometer measurements, and tend to lead to faster convergence (and less time delay when quick movements take place).

A variety of tests were run with different `beta` values and select results are detailed below. The Fo-SHIP was moved to different set points and the orientation was measured over time using Euler angle representation (converted from the Madgwick filter output quaternion). For future testing, it would be recommended to save and plot the set points in the same graph as the orientation estimate results to better verify the accuracy of the filter.

For Figures 4.14 through 4.17, the Fo-SHIP followed this list of instructions:

- Start at zero for all axes
- Move to roll = 24° over 1.2 seconds
- Hold position for 0.8 seconds
- Move to roll = 0° over 1.2 seconds
- Hold position for 0.8 seconds
- Move to pitch = 24° over 1.2 seconds
- Hold position for 0.8 seconds
- Move to pitch = 0° over 1.2 seconds

Values of `beta` above 0.1 produced very noisy results, and a value of 0.05 produced significant drift over time. This is expected: large values of `beta` place lots of trust in the accelerometer and magnetometer measurements, which are very noisy when moving; and small values place more trust in the gyroscope measurements, which have non-zero noise when stationary that produces drift over time. Drift in the yaw axis was minimal for large `beta` values and significant for small values.

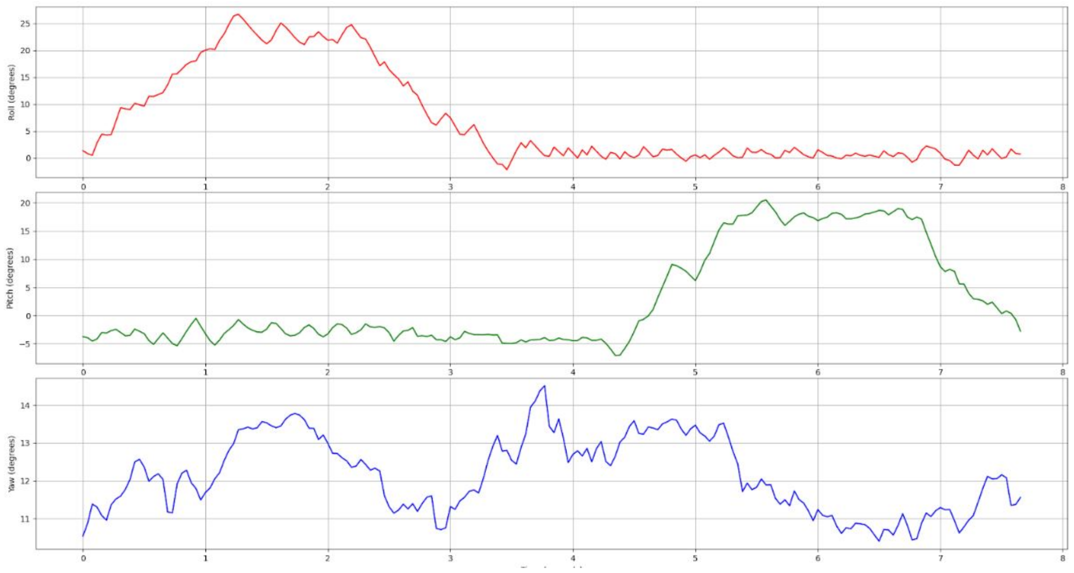


Figure 4.14: `beta = 0.3`, roll and pitch test with holds between

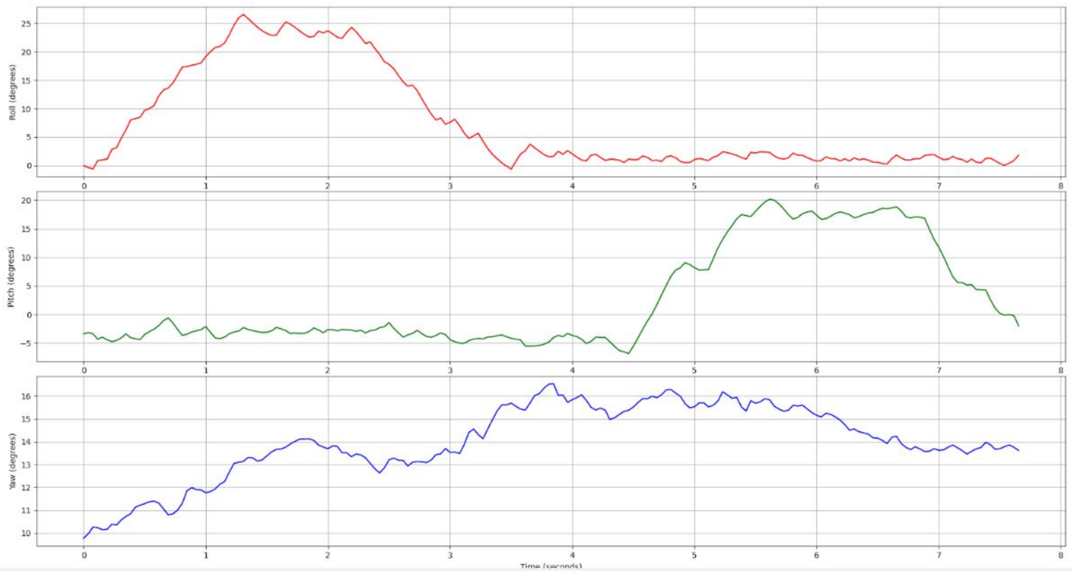


Figure 4.15: `beta = 0.2`, roll and pitch test with holds between

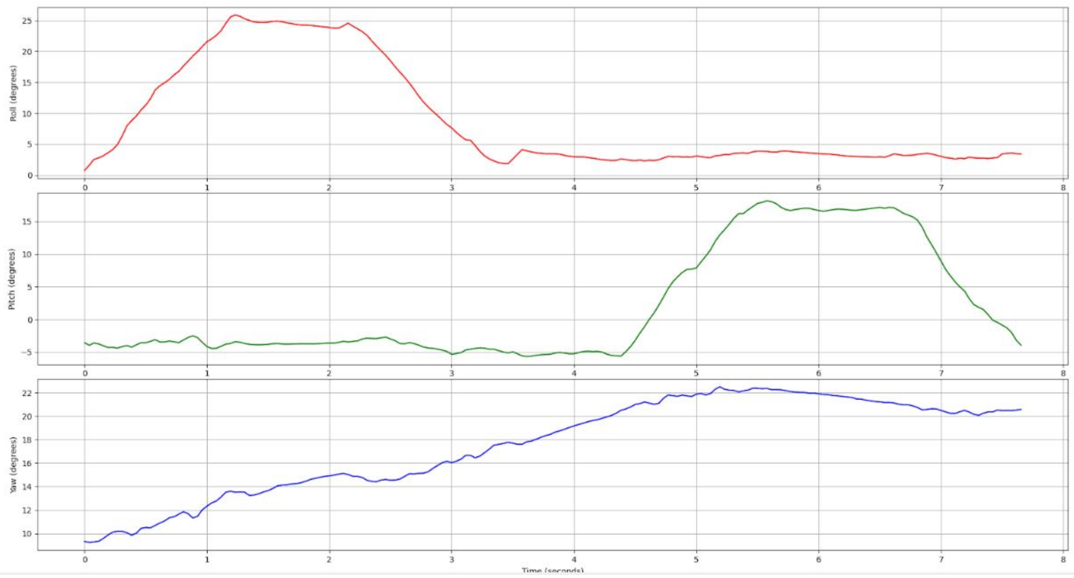


Figure 4.16: `beta = 0.1`, roll and pitch test with holds between

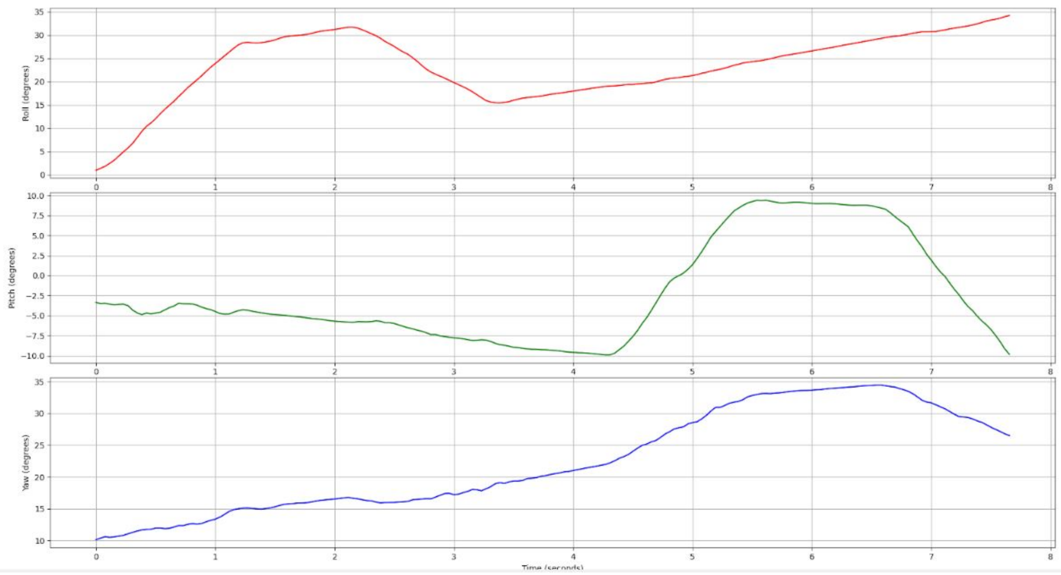


Figure 4.17: `beta = 0.05`, roll and pitch test with holds between

Due to the poor results of `beta = 0.3`, further tests were not run with this value. For Figures 4.18 through 4.20, the Fo-SHIP followed these movements:

- Start at zero for all axes
- Move to roll = 45° over 0.8 seconds
- Move to roll = -45° over 1.6 seconds
- Move to roll = 0° over 0.8 seconds
- Move to pitch = 45° over 0.8 seconds
- Move to pitch = -45° over 1.6 seconds
- Move to pitch = 0° over 0.8 seconds

The results of this test were much better than the previous test. The higher value of `beta` still produce noisier estimates for all angles, and the lower value still contained significant drift. The yaw values are positively correlated with the pitch values due to initial misalignment of the Fo-SHIP; if it were properly aligned with yaw = 0°, the yaw values would be much more constant.

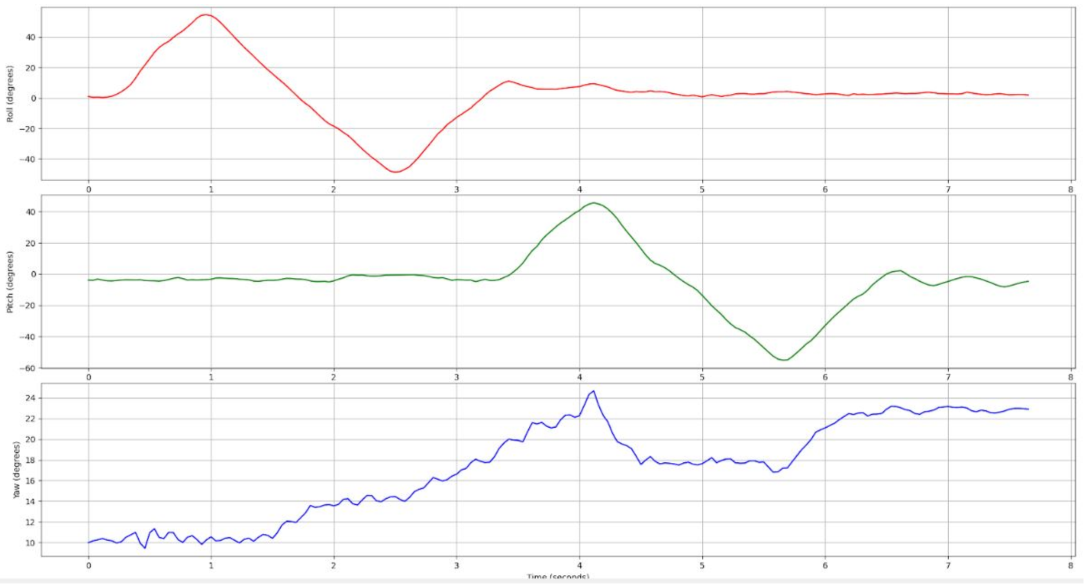


Figure 4.18: `beta = 0.2`, roll and pitch test with no holds

Note that the yaw component tends to be the least accurate across all the tests performed. This is to be expected; the magnetometer (which provides the absolute reference for yaw) is subject to many external magnetic fields (24 servo motors pulling up to five amps of current, power and data lines from the iSBL receivers placed near the IMU, etc.). Even when stationary, the Earth's magnetic field is quite weak compared to the non-moving electronics near the IMU.

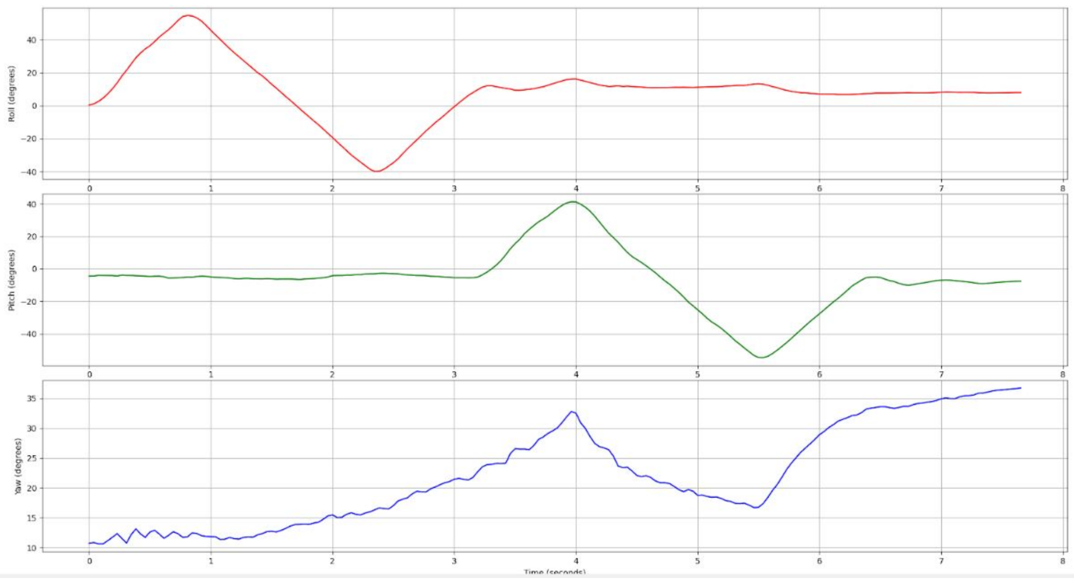


Figure 4.19: `beta = 0.1`, roll and pitch test with no holds

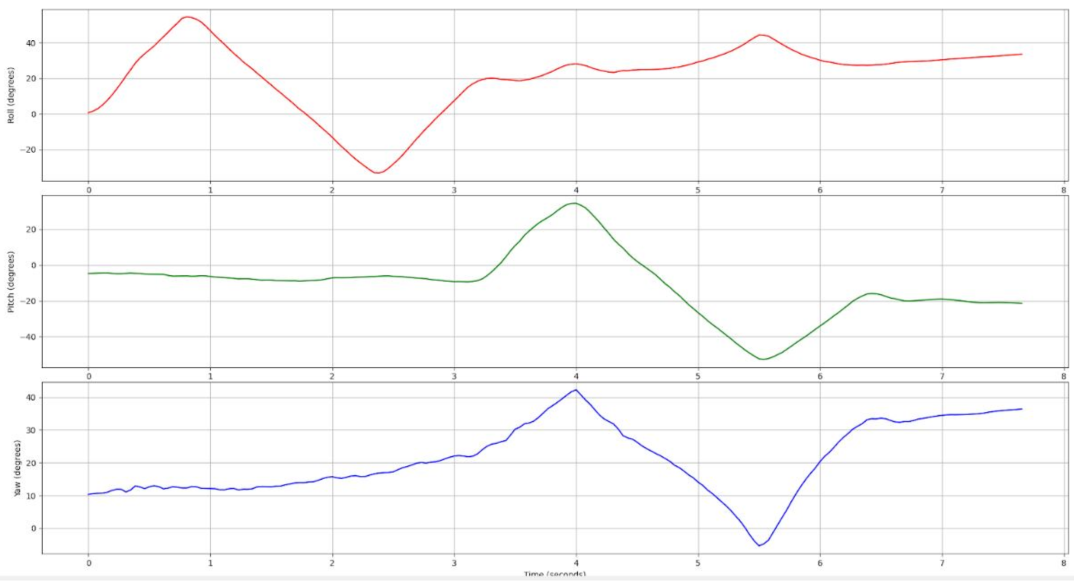


Figure 4.20: `beta = 0.05`, roll and pitch test with no holds

The value of `beta = 0.1` produced the least noisy orientation estimate over both tests while not drifting, so it was chosen as the starting point for the optimal `beta` value. After much tuning and testing, the value of `beta = 0.11` was found to give the best orientation estimate with the current implementation of the Madgwick filter. Figure 4.21 shows the results of a test where the maximum range of the Fo-SHIP was used: roll = ±60°, pitch = ±60°, and yaw = ±16°.

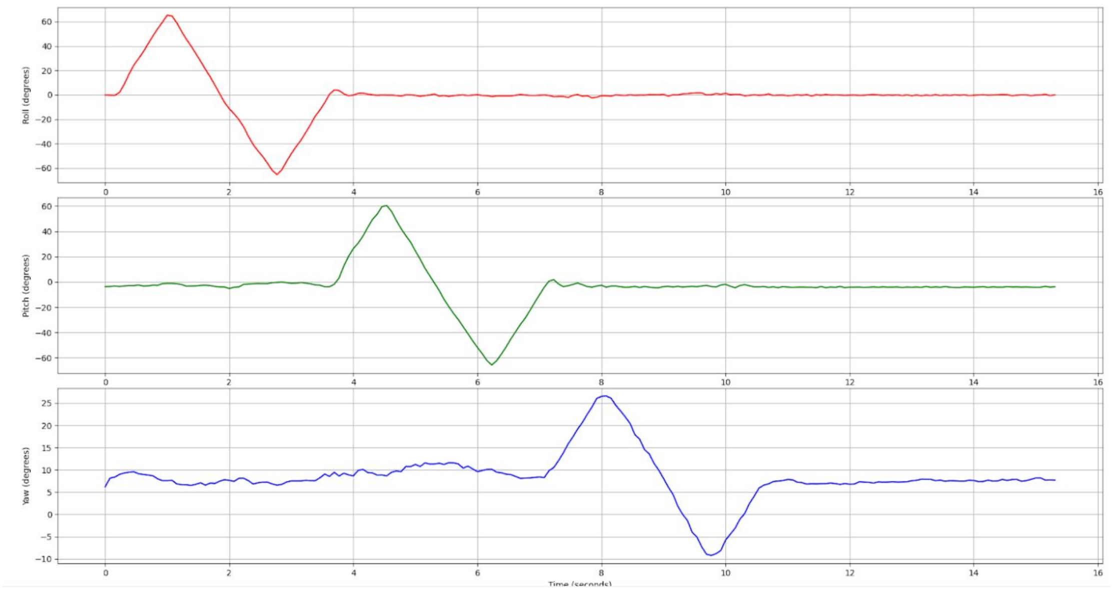


Figure 4.21: `beta = 0.11`, full Fo-SHIP range-of-motion test

Through these tests (and referencing similar implementations of sensor fusion filters running on consumer-grade IMUs [5] [6] [8] [19]), the steady-state accuracy of the orientation estimate was estimated for each axis of rotation; the results are shown below in Table 4.1. Note that this only holds for stationary orientation estimates; the accuracy of the orientation estimate during dynamic movement cannot be easily computed, and is estimated to be at least twice as worse as the steady-state accuracy. Also, note that the accuracies in Table 4.1 are only estimates. The Fo-SHIP does not have the positional accuracy to properly validate these numbers, and they were estimated using results from similar implementations as well as a few steady-state tests.

Table 4.1: Estimated accuracy of Madgwick filter orientation estimate

Euler angle representation	Estimated accuracy
Roll (about x-axis)	±1°
Pitch (about y-axis)	±1°
Yaw (about z-axis)	±2°

The dead reckoning position estimate was often off by a significant factor. The change in position given by the dead reckoning approach is fairly complex and has many sources of error:

- Incorrect orientation estimate (especially when moving) can result in incorrectly rotated acceleration vectors
- Small accelerations below the threshold were not incorporated into the motion of the platform

- Acceleration errors propagate twice through the position estimate (double integration)
- The constant velocity drift bias may be an oversimplification for the true velocity drift bias
- The update rate of the Madgwick filter (and the accelerometer) are relatively low compared to other dead reckoning systems [3] [8]

Table 4.2 shows some position estimates computed by the dead reckoning algorithm for z-axis movements only. The estimated change in position tends to be in the correct direction, but often undershoots the true magnitude of the movement. This is to be expected, given the assumptions made for the algorithm (see Section 4.5) and the sources of error above.

Table 4.2: True and estimated changes in position using dead reckoning

True Change in Position, mm (Fo-SHIP)			**Estimated Change in Position, mm (Dead Reckoning)**		
x	y	z	x	y	z
0	0	10	2.1	0	6.3
0	0	10	4.2	0.3	7.4
0	0	10	2.9	-0.9	5
0	0	-10	0.8	0.2	-5.2
0	0	-10	-1	0.3	-8.6
0	0	-10	-2.7	0.8	-7.9
0	0	30	6	0.6	20.1
0	0	30	-4.2	1.1	15.7
0	0	30	1.3	0	22.5
0	0	-30	2	-0.7	-10
0	0	-30	3.5	1.2	-26.3
0	0	-30	-2.1	-2.5	-20.5

Despite this inaccuracy, the dead reckoning measurements can give good insight into the motion of the iSBL array. When the array is stationary for multiple acoustic position estimates, the change in position given by the dead reckoning algorithm is zero; when it is moving between estimates, the change in position given is non-zero. This helps inform the Kalman filter model about the relative motion of the system and can result in better position estimates than the acoustic position estimate alone. The results and impact of the dead reckoning algorithm on the position estimate of the system are discussed in Chapter **??**.

Bibliography

[1] P. K. Allan, “A history of the ship’s compass,” *Naval History*, dec 2022. [Online]. Available: https://www.usni.org/magazines/naval-history-magazine/2022/december/history-ships-compass

[2] V. R. History, “How a gyroscope guides a rocket,” apr 2018. [Online]. Available: https://v2rockethistory.com/gyroscope-guides-rocket/

[3] A. Navigation, “Inertial measurement unit (imu) – an introduction,” jun 2024. [Online]. Available: https://www.advancednavigation.com/tech-articles/inertial-measurement-unit-imu-an-introduction/

[4] B. Sensortec, “Smart sensor: Bno055,” 2024. [Online]. Available: https://www.bosch-sensortec.com/products/smart-sensor-systems/bno055/

[5] S. O. Madgwick, “An efficient orientation filter for inertial and inertial/magnetic sensor arrays,” Tech. Rep., apr 2010. [Online]. Available: https://x-io.co.uk/downloads/madgwick_internal_report.pdf

[6] S. A. Ludwig, K. D. Burnham, A. R. Jimenez, and P. A. Touma, “Comparison of attitude and heading reference systems using foot mounted mimu sensor data: Basic, madgwick and mahony,” 2018. [Online]. Available: http://www.cs.ndsu.nodak.edu/~siludwig/Publish/papers/SPIE20181.pdf

[7] R. Mahony, T. AHamel, and J.-M. Pflimlin, “Nonlinear complementary filters on the special orthogonal group,” *IEEE Transactions on Automatic Control*, vol. 53, 2008. [Online]. Available: https://hal.science/hal-00488376/document

[8] S. A. Ludwig and K. D. Burnham, “Comparison of euler estimate using extended kalman filter, madgwick and mahony on quadcopter flight data,” in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 1236–1241. [Online]. Available: https://ieeexplore.ieee.org/document/8453465

[9] M. Morgado, P. Oliveira, and C. Silvestre, “Tightly coupled ultrashort baseline and inertial navigation system for underwater vehicles: An experimental validation,” *Journal of Field Robotics*, vol. 30, no. 1, pp. 142–170. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21442

[10] InvenSense, "Mpu-6000 and mpu-6050 product specification revision 3.4," 2013. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[11] Adafruit, "Adafruit lsm6dsox + lis3mdl - precision 9 dof imu - stemma qt / qwiic," 2024. [Online]. Available: https://www.adafruit.com/product/4517

[12] STMicroelectronics, "Lsm6dsox datasheet: 6-axis imu (inertial measurement unit) with embedded ai: always-on 3-axis accelerometer and 3-axis gyroscope," 2024. [Online]. Available: https://www.st.com/resource/en/datasheet/lsm6dsox.pdf

[13] ——, "Lis3mdl datasheet: Digital output magnetic sensor: ultralow-power, high-performance 3-axis magnetometer," 2023. [Online]. Available: https://www.st.com/resource/en/datasheet/lis3mdl.pdf

[14] A. Visconti and Albezanc, "lis3mdl platform independent driver," 2024. [Online]. Available: https://github.com/STMicroelectronics/STMems_Standard_C_drivers/tree/master/lis3mdl_STdC

[15] A. Visconti, C. Parata, and Albezanc, "lsm6dsox platform independent driver," 2024. [Online]. Available: https://github.com/STMicroelectronics/STMems_Standard_C_drivers/tree/master/lsm6dsox_STdC

[16] D. Tedaldi, A. Pretto, and E. Menegatti, "A robust and easy to implement method for imu calibration without external equipments," pp. 3042–3049, 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6907297

[17] M. Kok and T. B. Schon, "Magnetometer calibration using inertial sensors," *IEEE Sensors Journal*, pp. 5679–5689, 2016. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:719169/FULLTEXT01.pdf

[18] D. Rose, "Rotation quaternions, and how to use them," may 2015. [Online]. Available: https://danceswithcode.net/engineeringnotes/quaternions/quaternions.html

[19] x-io Technologies, "Open source imu and ahrs algorithms," jul 2012. [Online]. Available: https://x-io.co.uk/open-source-imu-and-ahrs-algorithms/

[20] T. S. Logsdon and J. L. Howard, "Dead reckoning," 2024. [Online]. Available: https://www.britannica.com/technology/navigation-technology/Dead-reckoning

[21] Y. Wu, X. Ta, R. Xiao, Y. Wei, D. An, and D. Li, "Survey of underwater robot positioning navigation," *Applied Ocean Research*, vol. 90, p. 101845, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141118718305546

[22] x-io Technologies, "Gait tracking with x-imu, script.m," 2017. [Online]. Available: https://github.com/xioTechnologies/Gait-Tracking-With-x-IMU/blob/master/Gait%20Tracking%20With%20x-IMU/Script.m