Institute for Cognitive Systems
Technische Universität München
Prof. Gordon Cheng

Humanoid Sensors and Actuators - Tutorial 6

Florian Bergner

Inertial Measurement Unit (IMU) (120 points)

In this tutorial we work with the Sparkfun 9DOF Razor IMU. This IMU integrates a 3D accelerometer, a 3D gyroscope and a 3D magnetometer. The accelerometer measures linear accelerations and the earth gravitational acceleration, the gyroscope measures angular velocities around the coordinate axes, and the magnetometer measures the magnetic flux of the earth magnetic field.

1 Accelerometer calibration (44 points)

To calibrate the accelerometer we have to find its offset and gain parameters. We assume that the coordinate axes of the accelerometer are orthogonal to each other and that the axes are aligned to the sensor module. To get the calibrated measurements we can use the following model:

$$\mathbf{x}_{\text{calib}} = \mathbf{M}\ (\mathbf{x}_{\text{raw}} - \mathbf{w}) \qquad \text{with} \quad \mathbf{M} \in \mathbb{R}^{3\times3}, \quad \text{and } \mathbf{x}_{\text{calib}},\ \mathbf{x}_{\text{raw}},\ \mathbf{w} \in \mathbb{R}^{3\times3} \tag{1}$$

The model matrix $\mathbf{M}$ is a diagonal matrix with gain values on its diagonal:

$$\mathbf{M} = \text{diag}(g_{\text{x}},\ g_{\text{y}},\ g_{\text{z}}) \tag{2}$$

To determine $\mathbf{M}$ and $\mathbf{w}$ we can use the ellipsoid equation since the raw sensor data points lie all on an ellipsoid as long as the linear accelerations are kept close to zero. We use the formulas and conventions introduced in lecture L5.

a) Setup

- Download the template project for the IMU and compile:
  `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T6/Material/ImuTemplate.tar.gz`
- Fetch an IMU unit, connect it to an USB cable and connect it to the PC.
- Run the IMU template program and fetch a set of sensor samples.

b) Getting started (4 points)

T.1.1 **(2 points)** How do we have to choose the general ellipsoid matrix $\mathbf{A}_{\text{fit}}$ such that the model matrix $\mathbf{M}$ becomes a diagonal matrix?

T.1.2 **(2 points)** What parameters ($A,\ B,\ \ldots,\ K$) of the approximated ellipsoid matrix $\tilde{\mathbf{A}}$ become zero?

c) Matlab implementation (16 points)

First we implement the accelerometer calibration algorithm in Matlab using a set of captured sensors samples. The Matlab implementation will be the reference for our online implementation in C++.

T.1.3 (4 points) Use the Matlab script `main_accCalib.m` which you can find in the IMU project folder and implement a least squares fitting algorithm to fit an ellipsoid to your measured data points. You solve an over-defined equation system using SVD. Hand-in your values for the approximated ellipsoid matrix $\tilde{\mathbf{A}}$ and the shift vector $\tilde{\mathbf{b}}$.

T.1.4 (4 points) Use your results of **T.1.3** and extend the Matlab script to find the ellipsoid offset $\mathbf{w}$. Hand-in your values for $\mathbf{w}$.

T.1.5 (4 points) Use your results of **T.1.3** and **T.1.4** and extend the Matlab script to find the general ellipsoid matrix $\mathbf{A}_{\text{fit}}$. Hand-in your values for $\mathbf{A}_{\text{fit}}$.

T.1.6 (4 points) Finally, use your results of **T.1.5** and extend the Matlab script to determine the model matrix $\mathbf{M}$. Hand-in your values for $\mathbf{M}$.

d) C++ Implementation (18 points)

Now we implement the accelerometer calibration algorithm in C++ using the Eigen math library. The quick reference guide for Eigen can be found in:
`https://eigen.tuxfamily.org/dox/group__QuickRefPage.html`
You can find further instructions in the classes *CalcAccCalib* and *ApplyAccCalib*.

T.1.7 (4 points) Finish the missing parts in the function `CalcAccCalib::solve()`. Use the following object to compute the SVD: `JacobiSVD<MatrixXd> svd(A,ComputeThinV | ComputeThinU);`

T.1.8 (4 points) Finish the missing parts in the function `CalcAccCalib::calcEllipsoidParameters()`. Use the function `CalcAccCalib::solve()` to solve linear equation systems.

T.1.9 (4 points) Finish the missing parts in the function `CalcAccCalib::calcEllipsoidCenter()`. Use the function `CalcAccCalib::solve()` to solve linear equation systems.

T.1.10 (2 points) Finish the missing parts in the function `ApplyAccCalib::applyCalib()`. Use the following object to compute the SVD: `JacobiSVD<Matrix3d> svd(m_A,ComputeFullV);`

T.1.11 (2 points) Finish the missing parts in the function `ApplyAccCalib::newRawImuData()`.

T.1.12 (2 points) Hand-in the correct values for $\mathbf{M}$, and $\mathbf{w}$. These values will be printed out at the console.

e) Report (6 points)

R.1.1 (2 points) How many different poses do we need at least to get a good set of parameters for the implemented calibration algorithm?

R.1.2 (2 points) Why should linear accelerations be minimized while capturing data for the calibration? Explain and elaborate.

R.1.3 (1 points) How do you minimize linear accelerations?

R.1.4 (1 points) How do you minimize the influence of linear accelerations during calibration?

R.1.5 (4 points) Describe and explain the mathematical trick we use to get the model matrix $\mathbf{M}$ from a properly scaled ellipsoid matrix $\mathbf{A}_{\text{fit}}$. Explain! Copying the formulas from the lecture script is NOT enough.

2 Magnetometer calibration (30 points)

The magnetometer calibration follows the same algorithm as the accelerometer calibration. The difference is that the measurement matrix $\mathbf{M}$ is now not restricted to a diagonal matrix.

a) Matlab implementation (8 points)

T.2.1 (4 points) Use the Matlab script `main_magCalib.m` and re-implement **T.1.3** - **T.1.6**. Note: This is NOT just a copy-and-paste task. You have to think and you will notice that you have to consider small but important differences.

T.2.2 (2 points) Hand-in the screen shots of your plots and present that the potato of un-calibrated sensor points is transformed into a proper unit sphere.

T.2.3 (2 points) Hand-in the correct values for $\mathbf{M}$, and $\mathbf{w}$.

b) C++ Implementation (6 points)

T.2.4 (4 points) Finish the missing parts in the functions:

- `CalcMagCalib::solve()`
- `CalcMagCalib::calcEllipsoidParameters()`
- `CalcMagCalib::calcEllipsoidCenter()`
- `ApplyMagCalib::applyCalib()`
- `ApplyMagCalib::newRawImuData()`

Note: This is NOT just a copy-and-paste task. You have to think and you will notice that you have to consider small but important differences.

T.2.5 (2 points) Hand-in the correct values for $\mathbf{A}$, $\mathbf{M}$, and $\mathbf{w}$. These values will be printed out at the console.

c) Report (16 points)

R.2.1 **(4 points)** What are the physical reasons for the distortions we observe in raw magnetometer data? You will get 2 points per correct answer. Explain and elaborate. You will get no points for copy-and-paste answers from Wikipedia.

R.2.2 **(4 points)** What are soft iron and hard iron effects? Explain and elaborate. You will get no points for copy-and-paste answers from Wikipedia.

R.2.3 **(4 points)** For capturing samples for the magnetometer calibration we move the IMU around and get samples for as many points on the ellipsoid surface as possible. Why can we do this here while we had to be very careful NOT to move the IMU during the accelerometer calibration? Explain and elaborate.

R.2.4 **(4 points)** What are the limits of this magnetometer calibration? You will get 2 points per correct answer. Explain and elaborate.

3 Gyroscope calibration (46 points)

The gyroscope calibration is easier to perform. First we determine the offsets. Then we use the calibrated accelerometer to determine the gain factor.

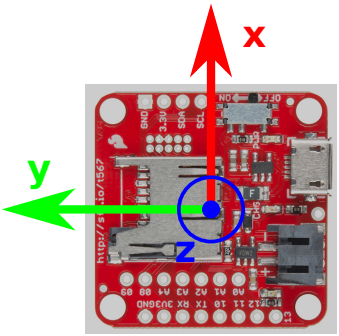

Figure 1: IMU with axes.

a) Implementation (24 points)

T.3.1 **(4 points)** Implement the on-line offset compensation in the IMU template C++ program. Finish the missing parts in the functions `CalcGyroCalib::calc()` and `ApplyGyroCalib::newRawImuData()`. Hand-in the correct values for $\mathbf{w}$.

T.3.2 **(16 points)** Inspect the `main.cpp` and adjust the coordinate axes orientations of all three sensors. Use the right hand rule for angular directions. See Fig. 1.

- **(4 points)** The axes of the accelerometer are correct. Hand-in ${}^{\text{new}}[\mathbf{T}_{\text{acc}}]_{\text{old}}$. Note, that you have to hand in also $\mathbf{M}$ of the accelerometer calibration since $\mathbf{T}_{\text{acc}}$ depends on $\mathbf{M}$, even if $\mathbf{M}$ is not correct.
- **(4 points)** The axes of the gyroscope are correct. Hand-in ${}^{\text{new}}[\mathbf{T}_{\text{gyro}}]_{\text{old}}$.
- **(4 points)** The axes of the magnetometer are correct. Hand-in ${}^{\text{new}}[\mathbf{T}_{\text{mag}}]_{\text{old}}$. Note, that you have to hand in also $\mathbf{M}$ of the magnetometer calibration since $\mathbf{T}_{\text{mag}}$ depends on $\mathbf{M}$, even if $\mathbf{M}$ is not correct.

T.3.2 **(4 points)** Determine the gain factor for the gyroscope (same for all three axes) using the Matlab script `main_gyroScale.m`. Hand-in the gain factor.

b) Report (22 points)

R.3.1 **(4 points)** Explain how a calibrated accelerometer can be used to find the gain factors for a gyroscope. Elaborate, derive and provide formulas.

R.3.2 **(6 points)** Which angles (roll, pitch, yaw) can be determined using only

- **(2 points)** the accelerometer,
- **(2 points)** the gyroscope, or
- **(2 points)** the magnetometer?

Explain and elaborate. Provide formulas.

R.3.3 **(6 points)** What are the drawbacks of

- **(2 points)** the accelerometer,
- **(2 points)** the gyroscope, and
- **(2 points)** the magnetometer?

Explain and elaborate. Consider physical impacts and noise. Just answering that a sensor cannot measure one or more of the angles is NOT sufficient.

R.3.4 **(4 points)** What sensor fusion algorithms can be used to improve the estimation of roll, pitch, yaw angels. You get 2 points for each mentioned algorithm. Please briefly explain in your own words how these algorithm work.

R.3.5 **(2 points)** Is it possible that these fusion algorithms outperform the individual sensors? If so explain and elaborate how.

Robot Skin (28 points)

In this part of the tutorial we want to introduce the robot skin we develop at ICS. This robot skin is a multi-modal skin which can measure temperature, accelerations, proximity and forces. First, we will use the integrated RGB LED of the skin cells to give the user feedback about different touch events. Then, we will use data of the accelerometer to determine the orientation between two skin cells.

4 Color feedback for touch events (8 points)

a) Setup

- Download the template project for the Skin and compile:
  `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T6/Material/SkinTemplate.tar.gz`
- Fetch a skin patch, connect it to the UART cable, and connect the FTDI UART/USB converter to the PC.
- Run the Skin template program.

b) Implementation (4 points)

T.4.1 (4 points) Follow the instructions in the template file and implement the color feedback in the function `LedFeedback::update()`. Try different thresholds and change the LED color according to the instructions.

c) Report (4 points)

R.4.1 (4 points) Why is providing feedback to the user important, especially in interaction tasks? Explain and elaborate.

5 Surface reconstruction and Orthogonal Procrustes problem (20 points)

The Procrustes problem is a matrix approximation problem where we want to find an orthogonal matrix $\mathbf{R}$ which maps a matrix $\mathbf{A}$ to a matrix $\mathbf{B}$. We can use the solution of the Procrustes problem to determine a rotation matrix $\mathbf{R}$ between two sets of vectors in matrices $\mathbf{A}$ and $\mathbf{B}$. To get a rotation matrix, the mapping between $\mathbf{A}$ and $\mathbf{B}$ needs to be constrained to rotation matrices. This can be enforced by forcing the determinant of $\mathbf{R}$ to 1. We can calculate the rotation matrix in the following way:

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \dots & \mathbf{a}_n \end{bmatrix} \in \mathbb{R}^{3\times n} \qquad \mathbf{B} = \begin{bmatrix} \mathbf{b}_1 & \dots & \mathbf{b}_n \end{bmatrix} \in \mathbb{R}^{3\times n} \tag{3}$$

$$\mathbf{M} = \mathbf{A}\,\mathbf{B}^T \in \mathbb{R}^{3\times 3} \qquad \mathbf{M} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^T \tag{4}$$

$$\hat{\mathbf{\Sigma}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \operatorname{sign}\left(\det\left(\mathbf{U}\,\mathbf{V}^T\right)\right) \end{pmatrix} \tag{5}$$

$$\mathbf{R} = \mathbf{U}\,\hat{\mathbf{\Sigma}}\,\mathbf{V}^T \tag{6}$$

a) Implementation (12 points)

T.5.1 **(12 points)** Follow the instructions in the template file and implement the Procrustes algorithm in the marked sections of function `Recon::sampledData()` using the math library Eigen:

- **(4 points)** The samples are normalized and added correctly to the matrices $\mathbf{A}$ and $\mathbf{B}$.
- **(4 points)** The Procrustes algorithm is implemented correctly.
- **(4 points)** The orientation and position of skin cell 2 are determined correctly. The calculation considers the absolute orientation of skin cell 1, the positions of the connection ports with respect to the skin cell coordinate frames, and the absolute position of cell 1.

b) Report (8 points)

R.5.1 **(4 points)** How many poses are needed to reliably determine the rotation matrix? Elaborate and explain.

R.5.2 **(4 points)** How do the singular values relate to an under defined, partially defined, fully defined and over defined solution for the rotation matrix? Elaborate and explain. Just copying the results of a paper is NOT enough.

c) Bonus (12 points)

Provide the mathematical analytical prove for the result of the Procrustes algorithm for a deeper understanding. As discussed, the Procustes algorithm tries to determine the best rotation matrix $\mathbf{R}$ to map a set of vectors $\mathbf{A}$ to another set of vectors $\mathbf{B}$. This breaks down to the following optimization problem:

$$\mathbf{R} = \underset{\mathbf{\Omega}}{\operatorname{argmin}} \|\mathbf{\Omega}\,\mathbf{A} - \mathbf{B}\|_{\mathrm{F}}^2 \quad \text{s.t.} \quad \mathbf{\Omega}^T\mathbf{\Omega} = \mathbf{I} \tag{7}$$

The Frobenius norm is defined as follows:

$$\|\mathbf{X}\|_{\mathrm{F}}^2 = \langle \mathbf{X}, \mathbf{X} \rangle_{\mathrm{F}} = \operatorname{tr}(\mathbf{X}^T\,\mathbf{X}) \tag{8}$$

The analytical simplification of the optimization problem breaks down to a SVD with the same result as shown above. Please provide all the steps necessary to achieve this result. Elaborate and explain each step.