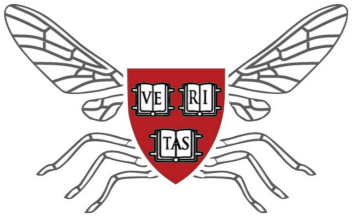
Harvard John A. Paulson
School of Engineering and Applied Science

and

Ecole Polytechnique Federale de Lausanne

A Proprioceptive Method for Soft Robots Using Inertial Measurement Units



Master's Thesis dissertation
April 22^{nd}, 2022

Author:
Yves Martin

Harvard Supervision:
Prof. R. J. Wood
Dr. D. K. Bruder

EPFL Supervision:
Prof. Mohamed
Bouri

Table of contents

1 Abstract

Proprioception, or the perception of the configuration of one's body, is challenging to achieve with soft robots due to their infinite degrees of freedom and incompatibility with most off-the-shelf sensors.

This work explores the use of inertial measurement units (IMUs), sensors that output orientation with respect to the direction of gravity, to achieve soft robot proprioception. A simple method for estimating the shape of a soft continuum robot arm from IMUs mounted along the arm is presented. The approach approximates a soft arm as a serial chain of rigid links, where the orientation of each link is given by the output of an IMU or by spherical linear interpolation of the output of adjacent IMUs. In experiments conducted on a 660mm long real-world soft arm, this approach provided estimates of its end effector position with a median error of less than 10% of the arm's length.

This demonstrates the potential of IMUs to serve as inexpensive off-the-shelf sensors for soft robot proprioception.

2 Introduction

2.1 Problem formulation

Proprioception, or the perception of the configuration of one's own body, is essential to robot planning and control. The configuration of a rigid-bodied robot can be fully described by a finite set of joint displacements which are readily measured using off-the-shelf sensors such as joint encoders. The configuration of a soft robot, however, is infinite dimensional and cannot readily be measured using off-the-shelf components.

2.2 State of the art

To address this shortcoming, a number of sensing technologies have been developed specifically for soft robots (1). Flexible resistive sensors infer strain by measuring the change in resistance of channels filled with conductive liquids such as liquid metals (2; 3) or ionic liquids (4) (Fig. 2.1). Flexible capacitive sensors estimate changes in geometry by measuring changes in capacitance of stretchable electrodes separated by an elastomeric dielectric layer (5). Optical strain sensors detect changes in geometry by measuring variations in intensity, frequency, or phase of light in a light transmission medium (6; 7; 8; 9) (Fig. 2.2). Magnetic strain sensors infer displacement by measuring the response of a Hall effect sensor embedded in a soft medium relative to a fixed magnetic field (10; 11). Inductive strain sensors estimate changes in geometry by measuring inductance variations caused by transducer mechanisms such as coil geometry and mutual inductance (12; 13; 14) (Fig. 2.3). Deformable sensing fabrics or "skins" that incorporate several soft sensing technologies into a single versatile package have also been developed (15; 16).

Each of these sensor types measure the strain of a flexible element. Using strain measurements to construct an estimate of the pose of a soft arm requires integrating strain along the length, imposing accuracy limitations since small inaccuracies in strain along the length will compound into much larger pose errors. Furthermore, while each sensor type has its own unique pros and cons, a common issue among them is nonlinear time-variant behavior and hysteresis. This has motivated the use of machine learning techniques to identify the mapping from raw sensor output to deformation from data (17; 18; 19).

An alternative to embedded sensors is an external motion capture system (Fig. 2.5). Such systems utilize an array of externally-mounted infrared cameras to track the position of reflective markers in 3D space. By coating a soft robot in reflective markers one can use motion capture data to estimate the robot's shape. Commercial motion capture systems offer an accurate and reliable method for sensing the deformation of soft robots, but they are expensive and impose severe restrictions on the environment

in which a robot can operate. Motion capture systems are not portable, are sensitive to lighting conditions, and are susceptible to errors due to occlusions, making it impossible to use them outside of a controlled laboratory setting.

This paper explores the use of Inertial Measurement Units (IMUs) for sensing the shape of continuum soft robots. IMUs are off-the-shelf orientation sensors that combine a three-axis accelerometer, three-axis gyroscope, and three-axis magnetometer with an on-board sensor fusion algorithm. Due to their extensive use in smartphones, tablets, and wearable fitness trackers, IMUs have become widely available and inexpensive, and there are many well-developed computational resources for integrating them into robotic systems (20).

Previous work has explored the use of IMUs for proprioception in soft robots. In (21), two IMUs were used in combination with motor encoders to estimate the pose of a single segment continuum body manipulator assuming constant curvature (Fig. 2.4a). In (22) two IMUs were embedded into the ends of an elastomeric liquid metal strain sensor to form a hybrid sensor capable of measuring the angle of deflection of a soft bending actuator and the joint position of a rigid robot arm (Fig. 2.4b). In both of these cases, IMUs were combined with other sensor types (i.e. encoders, strain sensors) to estimate the pose of a one segment soft structure.

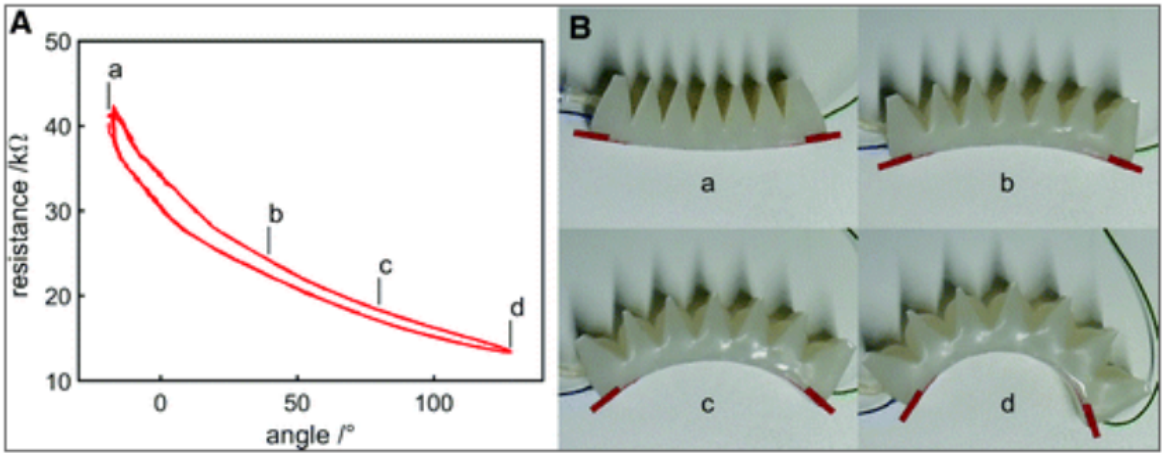


Figure 2.1: Helps and al., flexible fluidic sensor. (4)

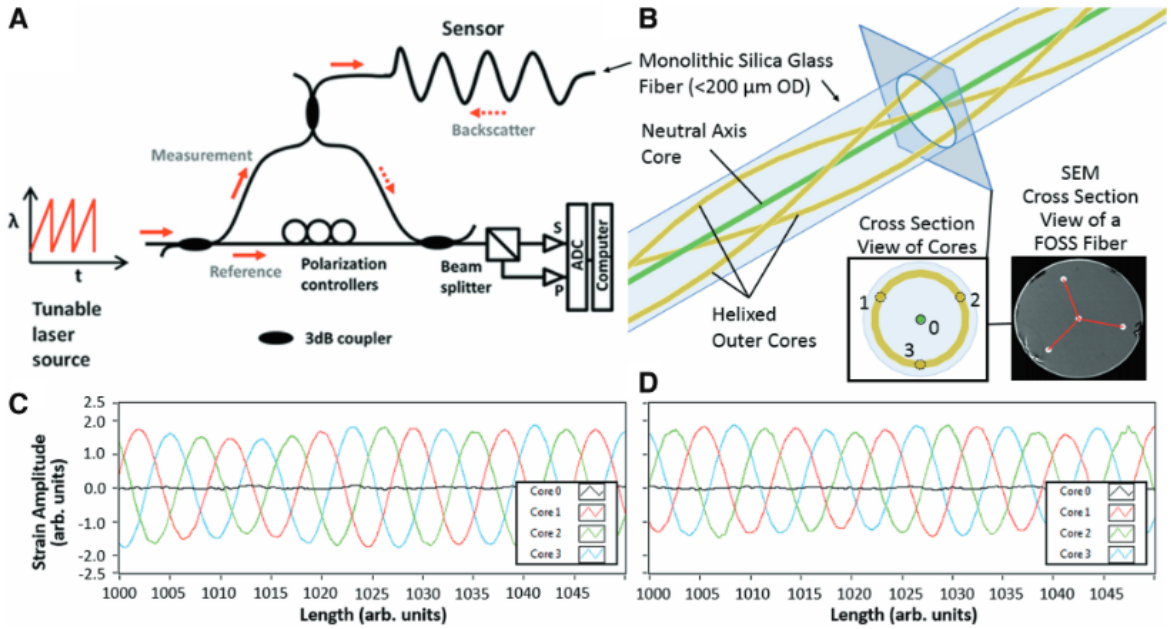


Figure 2.2: Galloway and al., fiber optic shape sensing for soft robotics. (6)

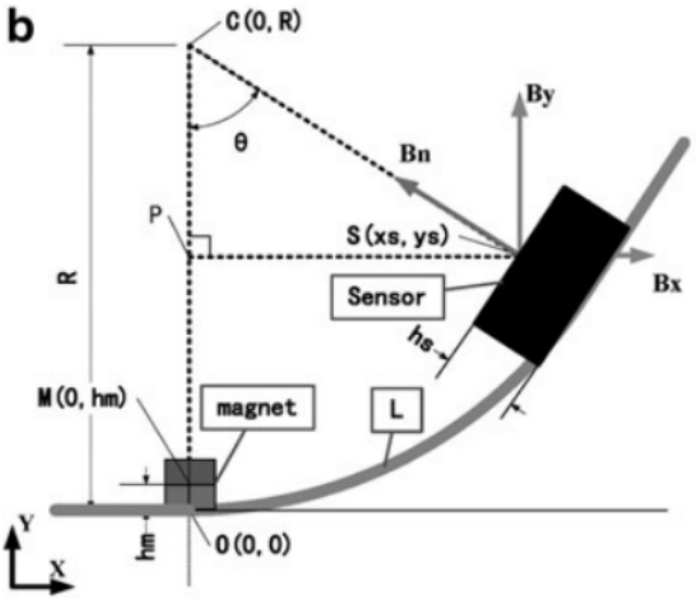


Figure 2.3: Luo and al., magnetic bending sensor. (10)

(a) Hughes and al., closed loop control of a constant curvature robot using an IMU and a motor encoder (21)

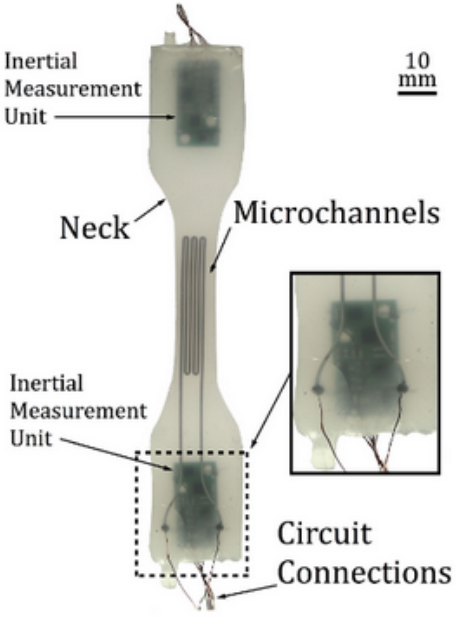


Fig. 1. A photograph of the prototype hybrid soft sensor

(b) Yirmibesoglu and al., fluidic strain sensor with embedded IMUs (22)

Figure 2.4: Existing soft-robotics sensing method using an IMU combined with another sensor.

2.3 Summary of contribution

The contribution of this master's thesis is a simple method for estimating the shape of a full continuum arm using only IMUs. The approach approximates the infinite-dimensional shape of a continuum arm with a finite-dimensional rigid-body model, and does not require any prior training. In real-world validation experiments, it is shown to estimate end-effector position with a median error of less than 10% of arm length. , putting its accuracy on par with that of other soft sensors while avoiding the limitations imposed by external motion capture systems.

2.4 Summary of content

This master's thesis is aimed at developing an embedded proprioceptive method to sense a soft-robotic full continuum arm using IMUs. A proper method has been developed, implemented using ROS, and tested using motion capture as a reference.

The state-estimation method consists in outfitting a soft arm with IMUs, and es-

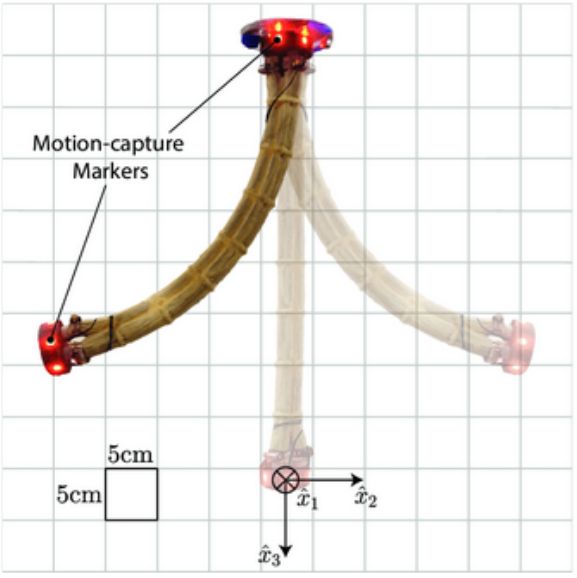


Figure 2.5: Motion capture markers placed at the base and the end-effector of a soft-robotic arm. (23)

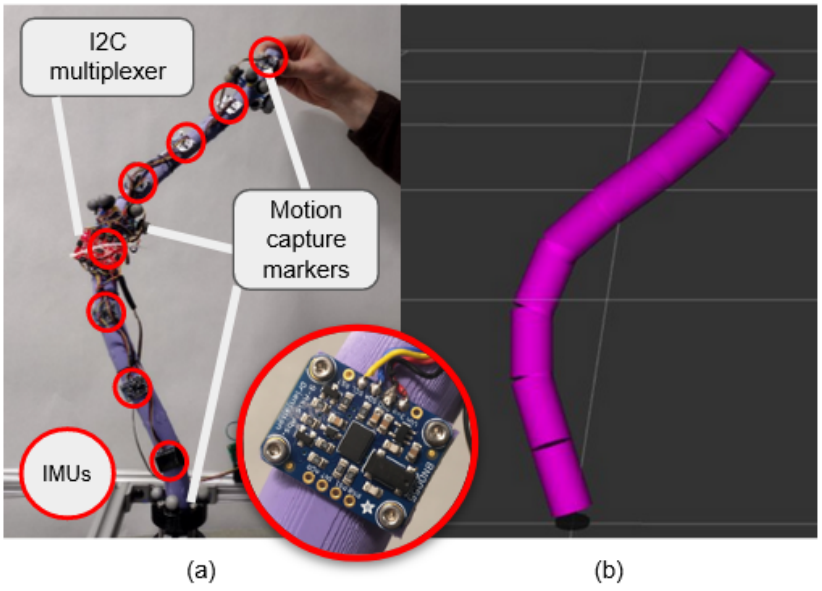


Figure 2.6: (a) The soft arm platform, with IMUs labeled, used to evaluate our estimation methods. (b) A serial-chain rigid-body model approximation of the soft arm motion is shown.

timating the state of the arm by interpolating the attitude measurements. To do so, we used different models to model the shape between two IMUs. One model represents these shapes as straight links, we call it the Rigid-body model. Another model represents these shapes as circular arcs, we call it the Piece-wise Constant Curvature (PCC) model. The PCC model exists in the literature, but under the Euler form only. To make the integration with the BNO055 - and computer science in general - easier, and to avoid the Gimbal lock, we developed a mathematical model to build the PCC model using quaternions. Then, to increase the performances, we created an simplified version of this model by replacing the circular arc by spherically interpolated fixed-length links, and quantified the performance of this approximation.

The experimental validation of the models was made using a real purely passive Silicon soft arm. We selected an inexpensive IMU, the Bosch BNO055, and we outfitted our arm with 8 of them. The BNO055 has an auto-calibration mode to mitigate the effect of the IMU drift, and it can be set to a previously saved calibration. The state estimation is implemented as a ROS package and runs in real time, displays the results through a visual interface, and records data. This ROS package is public, documented, and can be used with up to 8 IMUs and customised soft arms, to give direct access to this work to the research and the industry. Two motion capture markers groups were placed on the arm, almost at the middle and at the end-effector. Using the motion capture data as a reference, our method and implementation error was quantified.

The result of our experimentation is a median error in position around 10% for our best model. Whereas the error of the model increases along the arm, the error w.r.t the arm's length decreases along the arm. The IMU density has a strong impact on the accuracy. The PCC approximation using fixed-length links increases in accuracy as the amount of links increases, until it reaches a plateau effect of 16 links to model the entire arm. The main source of error (75%) is the non-PCC behavior, among which 4% could be related to the simplification using straight-links. The IMU-related errors account for around 17%, and the motion capture related error around 8%.

Compared to previous technologies such as strain sensors, our method performs in the same accuracy scale for a 15cm arm, i.e 10% of median error. Though, the fact that our method relies on absolute orientation sensors allows us recover much larger scale shapes than strain sensors, assuming PCC behavior. Another advantage is how inexpensive and available off the shelf IMUs are. This sensor is nevertheless limited by the IMU drift, and its sensitivity to magnetic field variations. The application can be the sensing of any shape and orientation of a soft body presenting a Piece-wise constant curvature behavior. The most obvious application is soft-continuum arms, but broader outreach is possible, such as proprioceptive VR suits.

In section 3 will be described the methods. In section 4 will be described the

experimentation. In section 5 will be described the results. In section 6 will be discussed the results. In section 7 will be concluded this master's thesis.

3 Modeling methods

IMU measurements provide the arm's orientation at given spots along the arm. This section will discuss how to use this data to estimate the full arm shape using two different arm models. One model represents the arm as a series of rigid segments, one per IMU, whereas the second one approximates a PCC behavior (Fig. 3.1). Both models will be described and explicit parametrizations will be given. To give a better understanding of the second model, the theory of the PCC model will be shown. A new method to compute the PCC model using quaternions have been developed and will be presented.

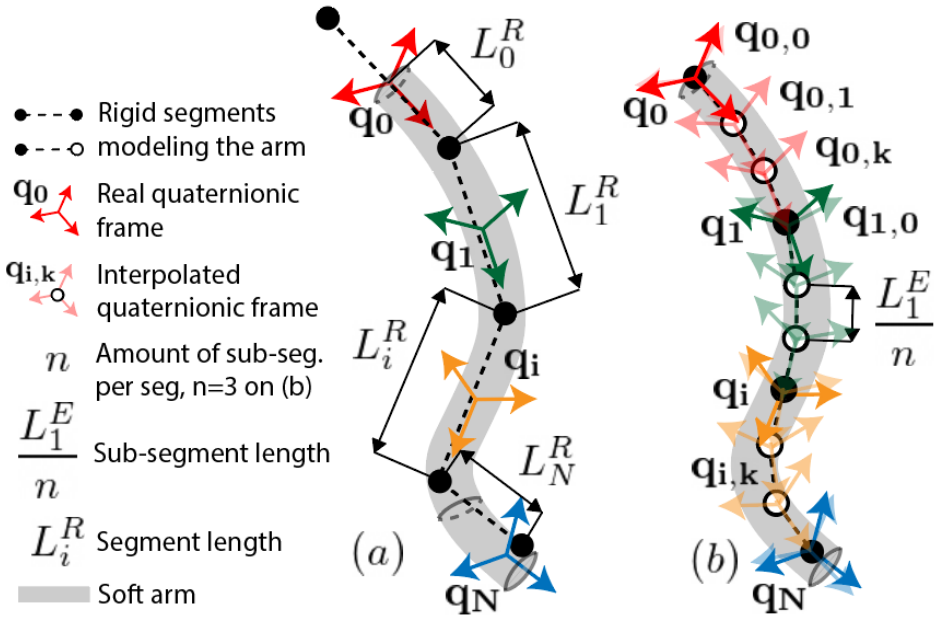


Figure 3.1: Different models to approximate the state of a soft continuum arm using IMUs: (a) rigid-body model, (b) and PCC-extended rigid-body model.

3.1 Model 1: the Rigid-body model

As a first approach to model the arm, we used a rough model. The arm is modeled by $N+1$ straight links of length L_i^R, $i \in \{0, ..., N\}$, called segments (see Fig. 3.1(a)). An IMU is fixed to each segment, ideally at the center. The i^{th} segment is oriented along the IMU frame defined by its quaternion $\mathbf{q_i}$, $i \in \{0, ..., N\}$ where $\mathbf{q_0}$ is the orientation of the arm's base. We define $R_q(\mathbf{q})$ as the 3x3 rotation matrix extracted from a quaternion $\mathbf{q} = a + ib + jc + kd$ (24),

$$R_q(\mathbf{q}) = \begin{pmatrix} 1-2(c^2+d^2) & 2(bc-ad) & 2(bd+ac) \\ 2(bc+ad) & 1-2(b^2+d^2) & 2(cd-ab) \\ 2(bd-ac) & 2(cd+ab) & 1-2(b^2+c^2) \end{pmatrix} \tag{3.1}$$

Let $\hat{\mathbf{x}}_\mathbf{i}^\mathbf{R}$ be an estimate of the position of the joint between segment i and segment $(i-1)$ according to this model. It is computed by summing the displacements of the first $i-1$ links, according to the following expression,

$$\hat{\mathbf{x}}_\mathbf{i}^\mathbf{R} = \sum_{j=0}^{i-1} R_q(\mathbf{q_j}) \begin{pmatrix} 0 \\ 0 \\ L_j^R \end{pmatrix} \tag{3.2}$$

Note that the superscript $(\cdot)^R$ denotes a variable related to the rigid-body model.

Note that this model can be optimized by computing only the 3rd column of the $R_q(\mathbf{q})$ for a given $\mathbf{q}$ since it's multiplicating a vector aligned with the z axis.

Note that the model specifies the shape of the arm as a collection of joint coordinates. However, because the model represents the arm as a series of rigid segments, it is straightforward to compute the coordinates of any point along the length of the arm using linear interpolation between two adjacent joints.

3.2 Model 2: the PCC-extended Rigid-body model

To make the rigid-body model smoother and more accurate, we divide each segment into n sub-segments. Each sub-segment orientation is computed under the assumption that the segment's curvature is constant.

As n increases, this model approximates the PCC model. The PCC model is broadly used in the soft robotics field, and consists in representing a soft arm as a series of circular arcs (in our case, each distance between two adjacent IMUs will be represented as a circular arc).

First, the PCC model will be presented, and described using Euler angles. Then, a new method to compute it using quaternions will be presented. The PCC-extended Rigid-body model will then be presented. Eventually, the fit between the PCC model and the PCC-extended Rigid-body model will be discussed.

3.2.1 Modeling a soft arm using the PCC model

In this method, we will show a theoretical model to estimate the shape of the arm. The input of this model is the measurements of the IMUs, i.e a vector of measurements of the orientation of the arm at different spots along the arm.

This model interpolates the orientation vectors by arcs of circle, assuming that the arm bends with constant curvature for each segment. A segment is here defined as a part of the arm with an IMU at each tip. Each arc of circle is determined by its length, i.e the length of the segment, and the IMUs orientation measurements as tangents at both tips.

On Fig. 3.2, one can see the constant curvature model for one segment. On Fig. 3.3, one can see the Piece-wise Constant Curvature model, i.e modeling the arm by several segments.

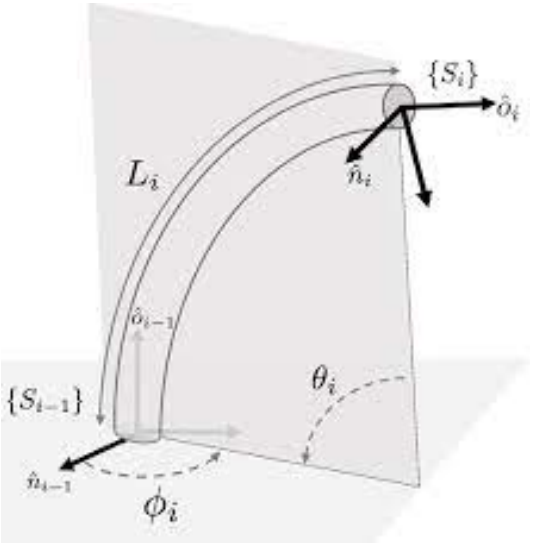


Figure 3.2: Constant curvature between two frames, modeling the behavior of a soft arm. (25)

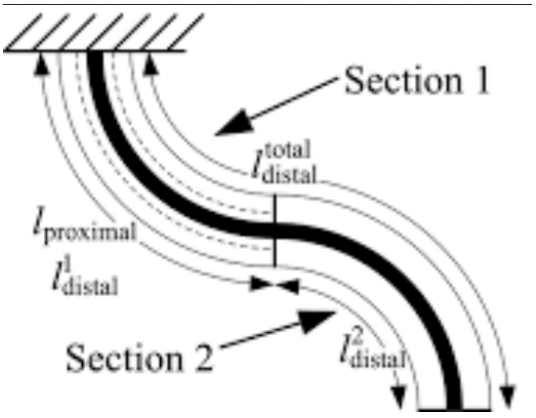


Figure 3.3: Piece-wise constant curvature model. (26)

The model's equations using Euler angles are shown on Fig. 3.6.

3.2.2 PCC mathematical formulation using quaternions

Since the output of the IMUs is a quaternion, we developed an expression of the PCC model presented in 3.2.1 using quaternions instead of Euler angles. This allowed us to avoid the Gimbal lock issue, to gain in computing efficiency, and to be easier to read. Quaternions are also broadly used in computer science and as a default IMU output format. It should be noted that we used here only purely imaginary unit quaternions.

First, we tried to find a PCC formula using vectors. We used the spherical linear interpolation (slerp function) shown in Fig. 3.4 to find an expression for a directional vector along the constant curvature segment. On the left, the vector $\vec{b}$, spherical linear

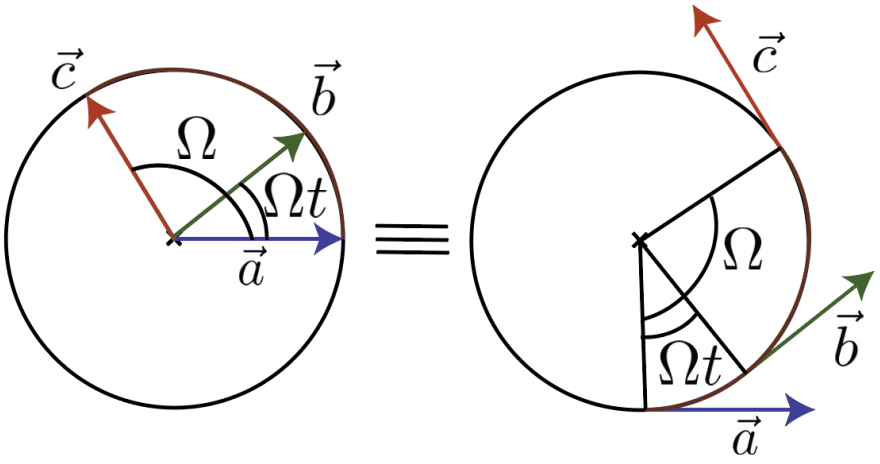


Figure 3.4: Spherical Linear Interpolation (slerp) between $\vec{a}$ and $\vec{c}$

interpolation between $\vec{a}$ and $\vec{c}$, is shown. On the right, the same vectors are placed differently, exhibiting how the integration of a circular arc in space using vectors can be done using spherical linear interpolation.

The spherical linear interpolation formula is the following:

$$Slerp(\vec{a}, \vec{c}, t) = \vec{b} = \vec{a} \cdot \frac{sin(\Omega(1-t))}{sin\Omega} + \vec{c} \cdot \frac{sin(\Omega t)}{sin\Omega} \tag{3.3}$$

By integrating this vector $\vec{b}$ along the interpolation sphere, we obtain a definition for the position vector such that $\vec{x(l)}$ is the position of the soft robotic spine at a distance l to the origin.

$$\vec{x}(l) = \int_0^l Slerp(\vec{a}, \vec{c}, \frac{l'}{L}) dl' = \int_0^l \vec{a} \cdot \frac{sin(\Omega(1-\frac{l'}{L}))}{sin\Omega} + \vec{c} \cdot \frac{sin(\Omega \frac{l'}{L})}{sin\Omega} dl' \tag{3.4}$$

Using the formal calculus software Wolfram Alpha, using two vectors $\begin{bmatrix} q11 \\ q12 \end{bmatrix}$ and $\begin{bmatrix} q21 \\ q22 \end{bmatrix}$, we found the equation 3.5. This solution was not satisfying in terms of bijectivity (Gimbal Lock effect), computational power and simplicity.

$$\left(\begin{array}{l} \frac{q_{21}\,\cos\left(\operatorname{acos}\left(\frac{q_{11}\,\overline{q_{21}}+q_{12}\,\overline{q_{22}}}{\sqrt{|q_{11}|^2+|q_{12}|^2}\,\sqrt{|q_{21}|^2+|q_{22}|^2}}\right)(t-1)\right)-q_{11}\,\cos\left(t\operatorname{acos}\left(\frac{q_{11}\,\overline{q_{21}}+q_{12}\,\overline{q_{22}}}{\sqrt{|q_{11}|^2+|q_{12}|^2}\,\sqrt{|q_{21}|^2+|q_{22}|^2}}\right)\right)}{\operatorname{acos}\left(\frac{q_{11}\,\overline{q_{21}}+q_{12}\,\overline{q_{22}}}{\sqrt{|q_{11}|^2+|q_{12}|^2}\,\sqrt{|q_{21}|^2+|q_{22}|^2}}\right)\sqrt{1-\frac{(q_{11}\,\overline{q_{21}}+q_{12}\,\overline{q_{22}})^2}{(|q_{11}|^2+|q_{12}|^2)\,(|q_{21}|^2+|q_{22}|^2)}}} \\ \frac{q_{22}\,\cos\left(\operatorname{acos}\left(\frac{q_{11}\,\overline{q_{21}}+q_{12}\,\overline{q_{22}}}{\sqrt{|q_{11}|^2+|q_{12}|^2}\,\sqrt{|q_{21}|^2+|q_{22}|^2}}\right)(t-1)\right)-q_{12}\,\cos\left(t\operatorname{acos}\left(\frac{q_{11}\,\overline{q_{21}}+q_{12}\,\overline{q_{22}}}{\sqrt{|q_{11}|^2+|q_{12}|^2}\,\sqrt{|q_{21}|^2+|q_{22}|^2}}\right)\right)}{\operatorname{acos}\left(\frac{q_{11}\,\overline{q_{21}}+q_{12}\,\overline{q_{22}}}{\sqrt{|q_{11}|^2+|q_{12}|^2}\,\sqrt{|q_{21}|^2+|q_{22}|^2}}\right)\sqrt{1-\frac{(q_{11}\,\overline{q_{21}}+q_{12}\,\overline{q_{22}})^2}{(|q_{11}|^2+|q_{12}|^2)\,(|q_{21}|^2+|q_{22}|^2)}}} \end{array} \right)' \tag{3.5}$$

To integrate the slerp function along the interpolation sphere, we tried another approach using quaternions. Quaternions are an extension of the complex numbers domain, and are generally represented under the form:

$$q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \tag{3.6}$$

where a, b, c, and d are real numbers; and i, j, and k are the basic quaternions. (27). In our situation, we will use purely imaginary quaternions, i.e $a = 0$ and $\vec{v} = \begin{bmatrix} b \\ c \\ d \end{bmatrix}$ where $\vec{v_i}$ is the normed vector of the orientation of the IMU i.

With quaternions, the equation for slerp shown in 3.3 is written:

$$Slerp(q_0, q_1, t) = (q_1 q_0^{-1})^t q_0 \tag{3.7}$$

Quaternions theory define an extensive amount of operations over quaternions, such as the product, the logarithm, the exponentiation, the reverse, and so on. Using these operations, we were able to integrate the slerp operation and to re-write the equation 3.4:

$$x(\vec{l}) = \int_0^l (q_1 q_0^{-1})^{\frac{l}{L}} q_0 dt = -L \cdot img\left((q_1 q_0^{-1})^{\frac{l}{L}} - 1)q_0\ log(q_1 q_0^{-1})^{-1}\right) \tag{3.8}$$

This representation avoids the Gimbal lock, is simpler, and requires less computational power. Using measured quaternions of the real system described further, a reconstruction of the arm is shown on Fig. 3.5.

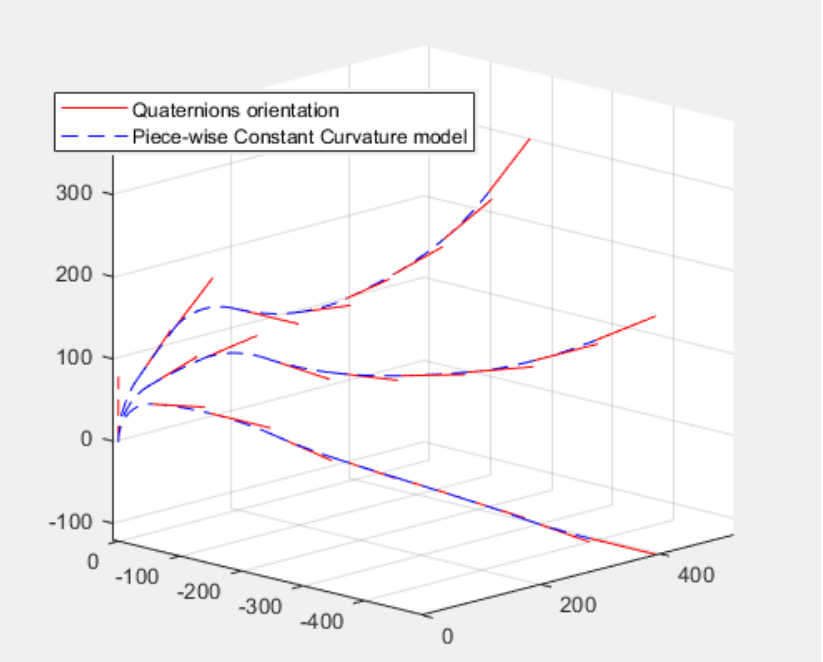


Figure 3.5: PCC model reconstruction using quaternions

$$T_w = \begin{bmatrix} \cos^2\phi(\cos\kappa s - 1) + 1 & \sin\phi\cos\phi\,(\cos\kappa s - 1) & \cos\phi\,\sin\,\kappa s & \frac{\cos\phi\,(1-\cos\kappa s)}{\kappa} \\ \sin\phi\,\cos\phi(\cos\kappa s - 1) & \cos^2\phi\,(1-\cos\kappa s) + \cos\,\kappa s & \sin\phi\,\sin\kappa s & \frac{\sin\phi\,(1-\cos\kappa s)}{\kappa} \\ -\cos\phi\,\sin\kappa s & -\sin\phi\,\sin\kappa s & \cos\kappa s & \frac{\sin\kappa s}{\kappa} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.6: PCC model using Euler angles. Θ and ϕ are shown in Fig. 3.2. κ is the curvature value. (25)

3.2.3 Approximating the PCC model using fixed-length links

The PCC model presented above is interesting as such, but heavy to compute if you have to compute every point along the arm. Therefore, we want to approximate this model using fixed-length links. This makes the model simpler and the computation easier, as well as easier to integrate in ROS. We will see here that fixed-length links can give an accurate representation of a circular arc as long as the number of links is calibrated w.r.t the arc. The most inaccurate point of this approximation is the end-effector, therefore, we will use the end-effector inaccuracy as an error reference.

This part will give a method that we developed to approximate circular arcs using

fixed-length links. An accuracy estimate of the end effector will be given w.r.t the arc's angle θ and the number of links.

The formula we used to approximate circular arcs using fixed-length links is to create K links of length $\frac{L}{K}$, where L is the length of the circular arc. The tangents at both arm tips are aligned with the vectors $\vec{v}_0^{tip}$ and $\vec{v}_1^{tip}$. The vectorial orientation of the link k is $\vec{v}_k^{link}$, $k \in \{0, ..., K-1\}$. See Fig. 3.7.

$$\vec{v}_k^{link} = slerp\left(\vec{v}_0^{tip}, \vec{v}_1^{tip}), \frac{2k+1}{2K}\right) \tag{3.9}$$

As we can see on Fig. 3.7, the end-tip position of the approximation will be aligned on the end-effector-to-origin axis, but not at the right distance since the length of the segments is fixed.

The approximation error w.r.t the number of segments is decreasing following the empirical law, cf Fig. 3.8:

$$Err(\theta, N_{seg}) = \frac{Err(\theta, 1)}{(N_{seg})^2} \tag{3.10}$$

$Err(\theta, 1)$ follows the curve shown on Fig. 3.9.

3.2.4 The PCC-extended Rigid-body parameterization and computation

Let us define L_i^E as the distance between $\mathbf{q_i}$ and $\mathbf{q_{i+1}}$, and $\mathbf{q_{i,k}}$ as the orientation quaternion of the k^{th} sub-segment of the i^{th} segment (see Fig. 3.1(b)), which is computed using spherical linear interpolation between $\mathbf{q_i}$ and $\mathbf{q_{i+1}}$.

$$\mathbf{q_{i,k}} = slerp\left(\mathbf{q_i}, \mathbf{q_{i+1}}, \frac{2k+1}{2n}\right) \tag{3.11}$$

The function *slerp* represents spherical linear interpolation (28) and is defined as,

$$slerp(\mathbf{q_a}, \mathbf{q_b}, t) = (\mathbf{q_b}{\mathbf{q_a}}^{-1})^t \mathbf{q_a} \tag{3.12}$$

where $t \in [0, 1]$ is the interpolation variable such that $slerp(\mathbf{q_a}, \mathbf{q_b}, 0) = \mathbf{q_a}$ and $slerp(\mathbf{q_a}, \mathbf{q_b}, 1) = \mathbf{q_b}$.

Next, the position of the k^{th} joint of the i^{th} segment, denoted $\hat{\mathbf{x}}_{\mathbf{i,k}}^{\mathbf{E}}$, is computed by summing the displacement of all preceding joints according to the following expression,

$$\hat{\mathbf{x}}_{\mathbf{i,k}}^{\mathbf{E}} = \hat{\mathbf{x}}_{\mathbf{i,0}}^{\mathbf{E}} + \sum_{m=0}^{k-1} R_q(\mathbf{q_{i,m}}) \begin{pmatrix} 0 \\ 0 \\ L_i^E/n \end{pmatrix} \tag{3.13}$$

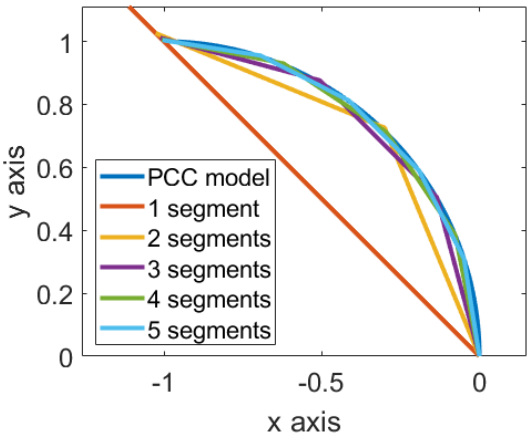


Figure 3.7: Approximation of an arc of circle of $\frac{\pi}{2}$ rad using 1 to 5 interpolation segments.

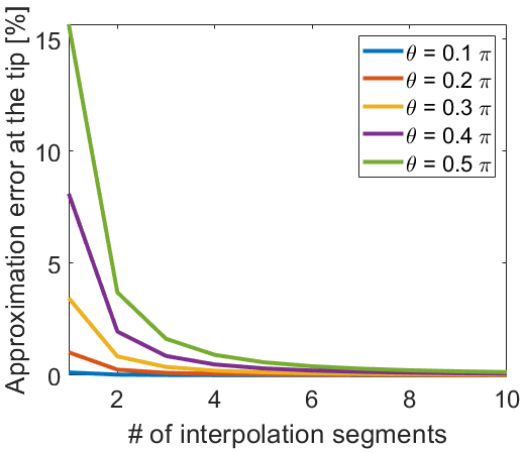


Figure 3.8: Approximation error w.r.t the number of segments.

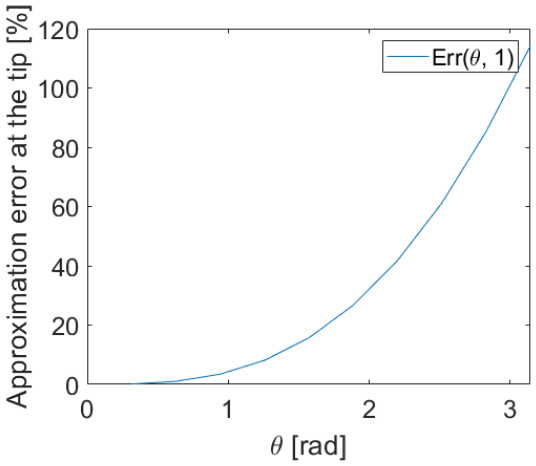


Figure 3.9: Error on the approximation of an arc of circle by a straight line in function of the angle θ of the arc.

Note that the superscript $(\cdot)^E$ denotes a variable related to the PCC-extended rigid-body model.

We define the 0^{th} joint of the 0^{th} segment to be located at the origin,

$$\hat{\mathbf{x}}^{\mathbf{E}}_{\mathbf{0,0}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.14}$$

and since the n^{th} joint of segment i is the same as the 0^{th} joint of segment $i+1$ the

following are equivalent,

$$\hat{\mathbf{x}}^{\mathbf{E}}_{\mathbf{i+1,0}} = \hat{\mathbf{x}}^{\mathbf{E}}_{\mathbf{i,n}} \tag{3.15}$$

4 Experimentation

In this part will be described how the performance of the IMU-based sensing approach was evaluated on a real-world soft arm platform. The features of this platform, of the electronics components, especially the IMU Bosch BNO055, and the architecture of the ROS software will be developed. Eventually, a description of the validation experiment will be given.

4.1 The soft arm platform

3 different prototypes have been developed, through an iterative process. The two first prototypes are shown here as an archive, the used prototype for experimentation is the prototype 3. Dimensions, shape, design and manufacturing details, photos of the prototypes, Fusion360 designs, IMU integration, motion capture system and integration will be shown and discussed for these prototypes.

4.1.1 Prototype 1

A first prototype was made out of rigid links and 2 IMUs (see Fig. 4.1a). Its purpose was to develop the ROS package on an easier prototype, and to evaluate the rigid-body model performance on a rigid-body structure.

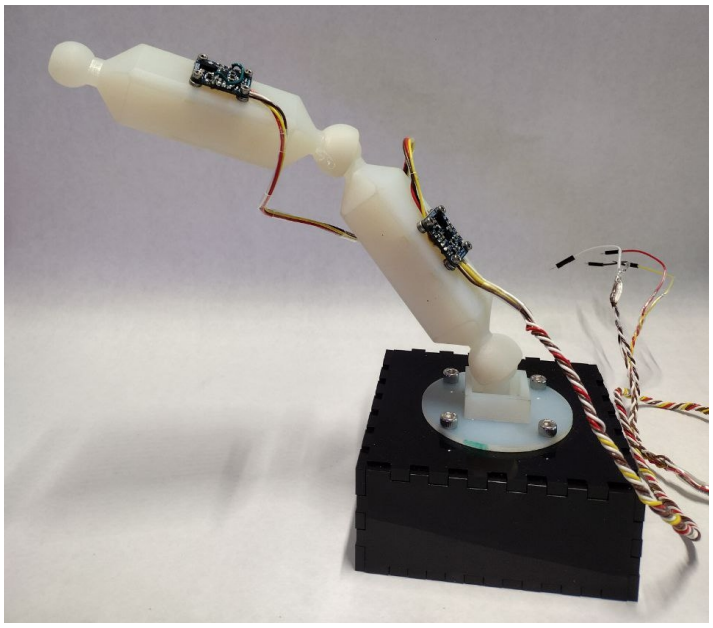
4.1.2 Prototype 2

A second prototype was made using flexible silicon joints and rigid bodies in between (see Fig. 4.2c). The joints are clamped to the rigid bodies using an intermediate rigid ring, and the two rigid parts are pressed against each other to hold the silicon tight. 8 BNO055 IMUs were used, using an I2C multiplexer. Silicon has to be clamped, otherwise it tears apart, which explains the designs of prototypes 2 and 3.

The goal of these prototypes was to experiment the rigid-body model. Though, we ran out of time and we wanted to focus more on the third prototype, which is more transferable to soft robots. This is why both prototypes have a ROS package developed to read them, but no experimental performance data.

4.1.3 Prototype 3

To validate the IMU-based modeling approach presented in Section 3, we constructed a soft continuum arm and outfitted it with IMU sensors. The arm is a cylinder of silicon (Smooth Sil 945) with a diameter of 25mm and of length 660mm. The structure is purely passive, it does not have any actuators. Eight flat slots were evenly spaced along the length of the arm and to allow the IMUs to be fixed in place with screws.



(a) 3D printed prototype with IMUs mounted on.

(b) Design on Fusion360

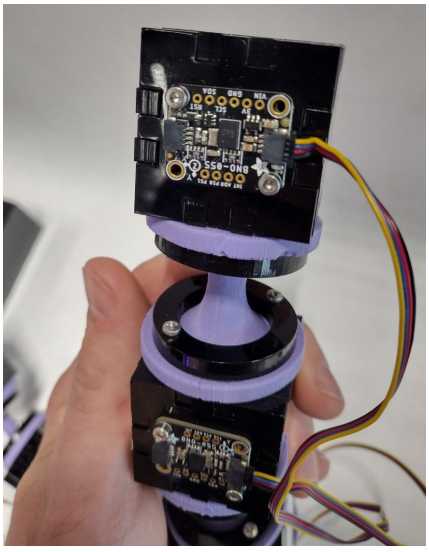
Figure 4.1: Prototype 1: two IMUs on 2 segments linked by a rotational joint.

We selected this geometry to make the arm as flexible as possible while accommodating the 25x20mm footprint of the IMU breakout boards and allowing them to be spaced 80-100mm apart. Empirically, this spacing permits a maximum angle of $\pi/2$ between adjacent IMUs.

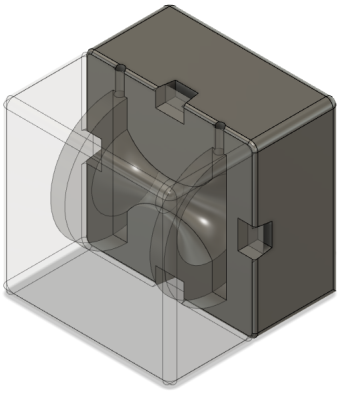
To manufacture this prototype, we created a mold shown on fig 4.3a. The mold was sprayed with an anti-adhesive for Silicon, such that the demolding would happen harmlessly. Then, we filled the mold with Smooth Sil 945, caring about not introducing bubbles in the mold. To avoid bubbles, the mold was hold in diagonal, such that the Silicon flows smoothly into the mold. Then the Silicon was put to cure for 2h in a 50°C oven. The arm was eventually demolded. This process was repeated once, in order to have two half of the final arm. The two parts were glued together.

4.1.4 The motion capture system

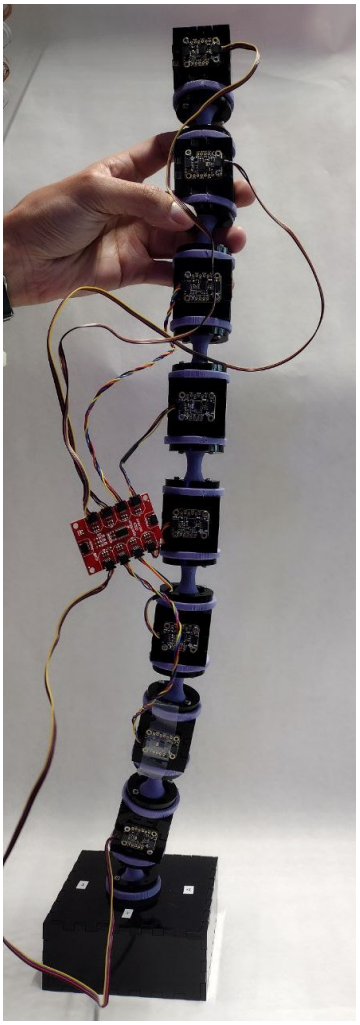
Two motion capture markers groups were fixed to the arm at lengths $M_1 = 360$mm and $M_2 = 600$mm. We call them Marker 1 and Marker 2. There are groups of markers for two reasons. First, it avoids occlusions, and second, the central symmetry pattern places the centroid of the markers group at the core of the arm. There was only two



(a) Detailed view of the flexible joint and its clamping to the rigid body.



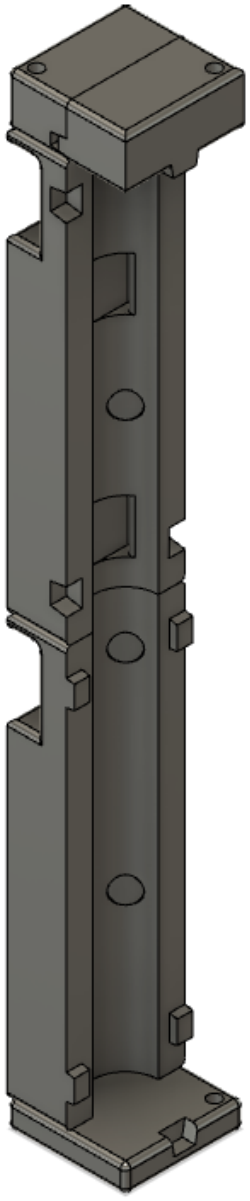
(b) Design of the joint mold on Fusion360.



(c) Prototype with IMUs mounted on.

Figure 4.2: Prototype #2: 8 IMUs on 8 segments linked by silicon flexible joints. The IMUs are read through an I2C multiplexer.

motion capture markers because when we tried to use a higher density of markers, the motion capture system would mix up the markers. The motion capture system (Vicon) has a precision of 1mm and an update frequency of 100Hz. The prototype 3 with motion capture markers is shown on Fig. 4.3b



(a) Half of the mold used to manufacture the soft arm.

(b) The real-world soft arm platform, with motion capture markers, without any electronics.

Figure 4.3: The prototype 3, i.e the soft arm platform on which the experiments were conducted.

Figure 4.4: Fixation of the arm's base.

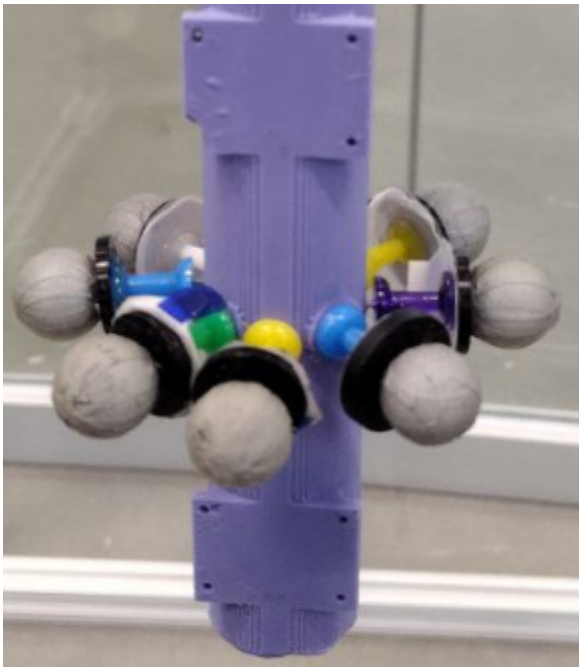


Figure 4.5: A motion capture markers group (here, the Marker 2).

4.2 IMU and Attitude and Heading reference systems (AHRS)

An inertial measurement unit (IMU) is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers (29). When the IMU outputs directly the orientation of the body using an embedded processor, it can be called an attitude and heading reference system (AHRS) (see Fig. 4.6).

In this section, we will present the Bosch BNO055, which we used as our IMU for the experimentation. Different sensor fusion techniques that exist to combine the measurements of the accelerometers, gyroscopes, and magnetometers into the absolute orientation of the sensor, will also be presented. The main issue with IMUs, i.e the drift, will be explained, justifying the need for calibration. The calibration method will be presented. As a bonus, a built-in error correction method that was developed but not used will also be presented.

4.2.1 The Bosch BNO055

For this project, we used the Adafruit BNO055 Absolute Orientation Sensor (30) shown in Fig. 4.7.

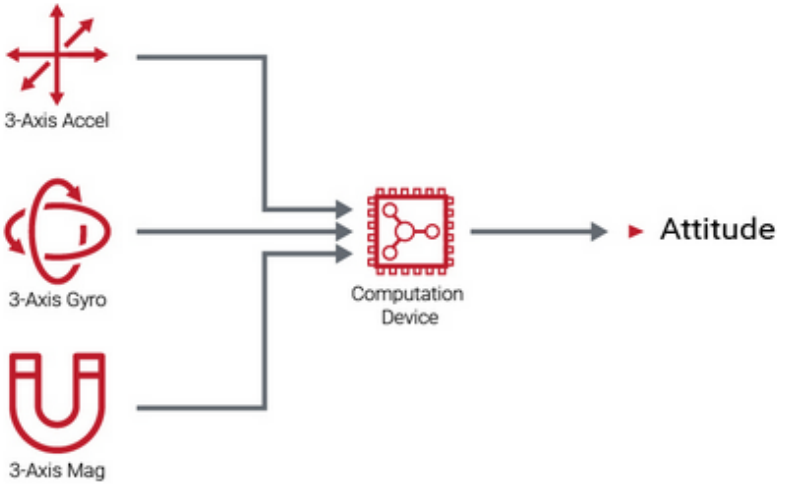


Figure 4.6: attitude and heading reference system (AHRS)

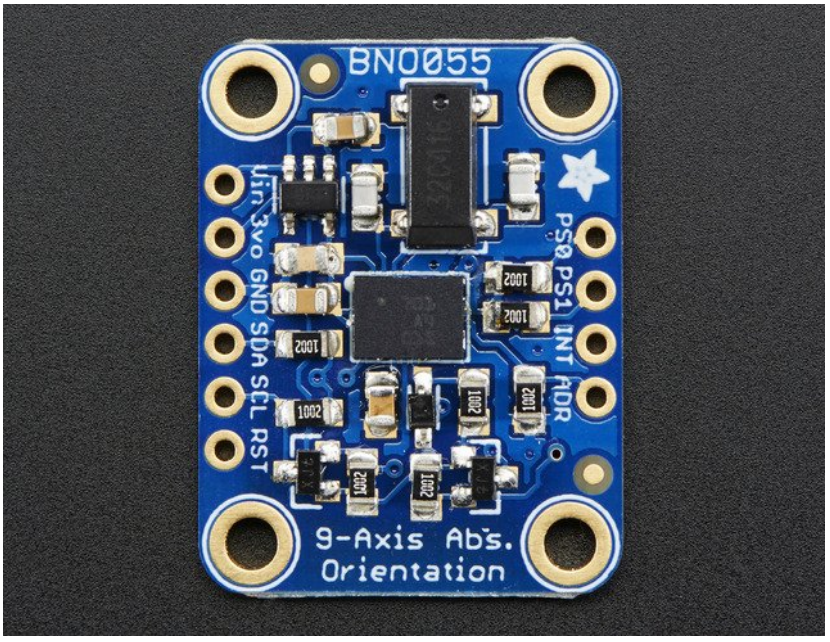


Figure 4.7: Adafruit BNO055 Absolute Orientation Sensor

The smart sensor BNO055 is a System in Package (SiP) solution that integrates a triaxial 14-bit accelerometer, an accurate close-loop triaxial 16-bit gyroscope, a triaxial geomagnetic sensor and a 32-bit microcontroller running the BSX3.0 FusionLib software. This smart sensor is significantly smaller than comparable solutions. By integrating sensors and sensor fusion in a single device, the BNO055 makes integration easy

(31), and provides orientation data under the form of quaternion. According to the Bosch BNO055 datasheet, reading a quaternion instead of the gyroscope, magnetometer, and accelerometer data makes the communication 10x faster. Plus a quaternion is easy to integrate into existing software frameworks such as ROS or Unity.

The basic description of the BNO055 is the following. The output rate of the orientation data is 100 Hz. Low latency is 20 ms. The sensor needs to be calibrated every time the power is set off, because there is no EEPROM memory on-board. Though, the sensor allows to save its calibration, and to load it into the sensor. The VDD voltage range is 2.4V to 3.6V. The sensor allows I2C and UART communication, and can operate between -40°C and +85°C. It is available off the shelf for less than 2$, which makes it usable for multiple-IMUs non-expensive applications.

According to Lin and al. (32), the accuracy of the BNO055 is the following:

Measurement	Roll error (deg)	Pitch error (deg)	Yaw error (deg)
Turning 90 deg	1.07	0.53	1.28
Turning 180 deg	0.74	0.73	2.51
Turning 270 deg	1.41	0.66	3.53
Turning 360 deg	0.44	0.86	4.61

Table 1: Table VI Dynamic test results of BNO055

Each of the sensors are programmable. The accelerometer can set a low pass filter (range 1kHz to 8Hz) and the acceleration range can be set to ±2g/±4g/±8g/±16g. The gyroscope range is switchable from ±125°/s to ±2000°/s, and a low-pass filter can be set with a bandwidth from 523Hz to 12Hz. Regarding the magnetometer, the resolution is 0.3µT. (31). This information wasn't directly used for our application, but is good to keep in mind.

4.2.2 Sensor fusion techniques

The main algorithms are Mahony's filter (33), Madgwick's filter (34) and NXP Sensor Fusion. The most accurate the algorithm is, the more it consumes computing power. Mahony's algorithm is a basic but effective algorithm that will run on smaller chips like the '328p which makes it a great one for any platform. Madgwick's algorithm is very popular when you have faster Cortex M0, M3, M4 or faster chips. It isn't going to run on an atmega328p. NXP Sensor Fusion is a really nice fusion algorithm was designed by NXP and requires a bit of RAM (so it isnt for a '328p Arduino) but it has great output results (35).

In this work, we used the BNO055 Bosch as our IMU. The BNO055 integrates a M0 processor running a custom Bosch sensor fusion algorithm (non open source),

which makes it an AHRS. Given that the sensor uses a M0 processor, it is allowed to think that the embedded algorithm is Madgwick-based. The performances of this sensor are <2% angle error, cf Tab. 1.

4.2.3 IMU drift and BNO055 auto-calibration feature

Gyroscopes are subject to bias instabilities, in which the initial zero reading of the gyroscope will cause drift over time due to integration of inherent imperfections and noise within the device. Bias repeatability can be calibrated across the known temperature range of the IMU. However, integrating constant bias instability will cause angular error. These errors will accumulate as gyroscope-based rotation or angle estimates drift over the long-term. The undesirable result of drift is that the error of a computed heading increases continuously unabated. Accelerometers, conversely, are sensitive to vibration and other non-gravity accelerations (36). Magnetometers are sensitive to changes in the surrounding magnetic field.

Because of the IMU drift presented above, we need to calibrate the sensors of the IMUs in order to have the most accurate results. Each sensor (gyroscope, magnetometer, accelerometer) needs to be calibrated independently. The BNO055 has a calibration routine that auto-calibrates the sensors continuously. The calibration routine provides an estimation of the IMU calibration state, for the system and for each sensor, ranging with a score from 0 to 3. (30). It should be noted that the yaw axis in particular is very sensitive to drift, and is estimated using the magnetometer.

4.2.4 IMU formal calibration

By achieving the following steps, the user can calibrate the IMU to the maximum score using the auto-calibration feature. We will call this step sequence and the generated calibration data "formal calibration". The calibration can be saved to a computer and loaded into an IMU. Since the BNO055 doesn't have any onboard EEPROM, it is very useful to formally calibrate the IMUs from time to time, and to load this calibration into the IMU at boot.

1. The calibration of the gyroscope is the easiest. The user only has to let the IMU steady for a few seconds.
2. The calibration of the magnetometer must be made by orientating the IMU with an 8 shape, to have it identifying the Earth magnetic field, like shown in Fig. 4.9.
3. The calibration of the accelerometer is the longest. The IMU must be hold steady for a few seconds, either one side of the IMU aligned with an horizontal surface, either forming a 45 deg angle with this flat surface.

To make the load and save calibration process easier, the ROS package includes a module to handle this.

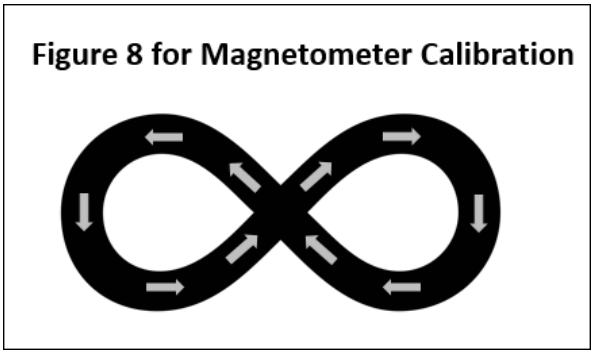


Figure 4.8: Magnetometer calibration pattern.

To make calibration faster, we manufactured a calibration bench. The point was to calibrate all of the IMUs at the same time by fixating them to the same surface. Also, since the bench is mounted on a rectangular parallelepiped, it is easy to orientate the IMU exactly along the right axis w.r.t the gravity using a spirit level. Though, this bench wasn't suitable because of the interference between the cables. The I2C protocol is sensitive to noise, and the IMUs kept resetting because of the interference, so we dropped the multi-IMUs calibration bench to use it with only one IMU.

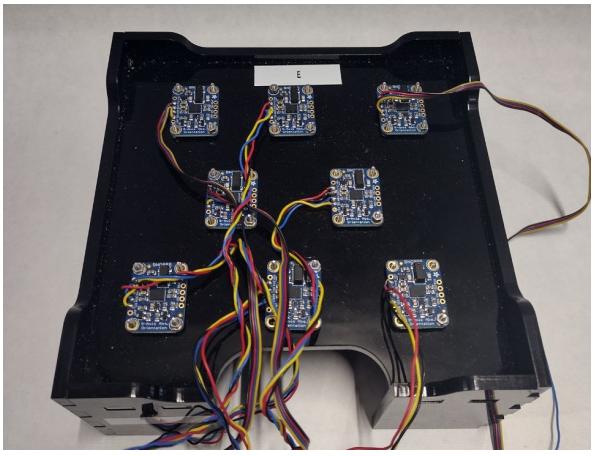


Figure 4.9: Calibration bench.

4.2.5 Accelerometer built-in errors correction

This part is not used in the final thesis, however, we found interesting to show this research on the IMU's accelerometer calibration. The idea is to remove bias, axis misalignment, and to re-scale the measurement, based on static measurements.

Based on the work of Ferrari and al. (37), we developed and tested computationally a method to correct built-in errors of the accelerometer. The goal was to remove the biases in the measurements along each axis, to realign the measurements with the axes of the IMU's frame (Fig. 4.10), and to scale the measurements. To do so, we measured the IMU in 6 given positions (Fig. 4.11), and then we compared it to the theoretical model (the gravity vector). Using mean-squares algorithm, we found the optimal scale, bias removal and axis realignment w.r.t to gravity measurement, to realign the accelerometer with its frame. For the records, each position was measured with the same amount of samples. The local gravity in Boston is found using (38).

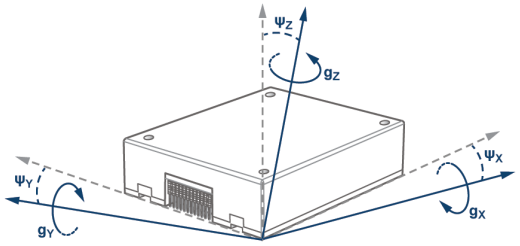


Figure 4.10: Misalignment of the IMU axis w.r.t the theoretical axis.

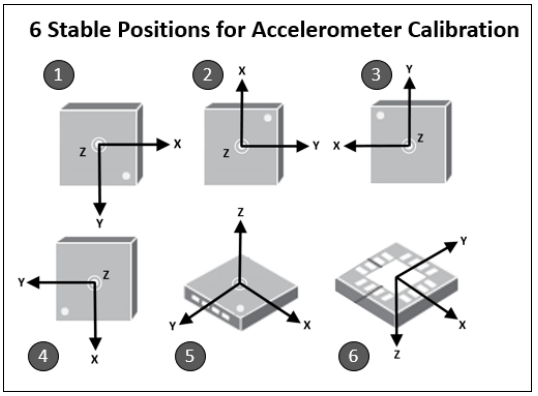


Figure 4.11: Proceeding for taking measures of the IMU in known positions, in order to calibrate it.

The mathematical model is the following:

$$b_{axis} = mean(\hat{S}^{+}_{axis}, \hat{S}^{-}_{axis}) \tag{4.1}$$

Where $\vec{b}$ is the bias, and $(\hat{S}^{+}_{axis}, \hat{S}^{-}_{axis})$ are measurements with the IMU oriented towards gravity w.r.t the target axis, resp. upwards and downwards.

$$T = \left((\hat{S} - \vec{b}) \cdot transpose(S)\right) \cdot (S \cdot transpose(S))^{-1} \tag{4.2}$$

Where T is the axis alignment matrix, computed using the least mean square algorithm, between the unbiased measurements $\hat{S} - \vec{b}$, and the theoretical axis model S.

Eventually, for a given measurement $\hat{s}$, we can compute an unbiased and realigned result $\hat{s}_c$ using the following equation:

$$\hat{s}_c = T \cdot (\hat{s} - \vec{b}) \tag{4.3}$$

Here is the MATLAB code that tested and validated our hypothesis:

```
% Constants
g = 9.80368; N = 1000;
sigx = 0.03; mux = -0.2;
sigy = 0.01; muy = 0.5;
sigz = 0.03; muz = -0.1;
sign = 0.03; mun = -0.6;

% Generate artificial data using Gaussian distribution
[S,Sb] = GenerateData(g, sigx, sigy, sigz, sign, mux, muy, muz, mun...
    , N)

% Use our calibration function
[outCal, TCal, BCal] = Calibration(S,Sb)

% Evaluate the performances over a large sample
i = 0; scoreCalibration = 0; scoreNoCalibration = 0;
while i<1000
    S = GenerateData(g, sigx, sigy, sigz, sign, mux, muy, muz, mun,...
    N);
    scoreCalibration = scoreCalibration + rms(Calibration(S,Sb)-Sb,...
    'all');
    scoreNoCalibration= scoreNoCalibration + rms(S-Sb, 'all');
    i = i+1;
end

% Show results
scoreCalibration
scoreNoCalibration
```

```
function [cal, T, B] = Calibration(S, Sb)
% Inputs:
%   S: Measurements of the IMU at a steady state in given positions
%   Sb: Theoretical value of the S measurements (w.r.t the IMU ...
    orientation and the gravity)
% Outputs
%   cal: function s.t if x is the output of an IMU, y = cal(x) ...
    gives y s.t the axis are aligned and unbiased
%   T: the axis alignment matrix
%   B: the bias vector

    % Find the biases matrix
```

```
    B = mean(S,2);

    % Solve T*S = Sb using LS algo
    T =((S-B)*Sb')\(Sb*Sb');

    % This is the final Calibration Function
    cal = @(s) T*(s-B);
end
```

```
function [S, Sb] = GenerateData(g, sigx, sigy, sigz, sign, mux, muy...
    , muz, mun, N)

% First part
Sb = zeros(3, 6*N); v = [ones(1,N) -ones(1,N)]*g;
for i = 1:3
    Sb(i,(i-1)*2*N+1:2*i*N)=v;
end

Rx = @(x) [1 0 0 ; 0 cos(x) sin(x); 0 -sin(x) cos(x)];
Ry = @(y) [cos(y) 0 sin(y); 0 1 0; -sin(y) 0 cos(y)];
Rz = @(z) [cos(z) -sin(z) 0 ; sin(z) cos(z) 0; 0 0 1];
S = Rx(normrnd(mux,sigx))*Ry(normrnd(muy,sigy))*Rz(normrnd(muz,sigz...
    ))*(Sb+normrnd(mun,sign,size(Sb)));
end
```

When adding Gaussian noise to the axes, with the values showed in the code above, we obtain a ScoreCalibration/ScoreNoCalibration value of around 1%, showing an evidence of significantly better results using this bias removal and axis realignment.

4.3 Electronics

4.3.1 The electronics schematics

In this part will be described the electronics. The BNO055 has been extensively described in part 4.2.1, so it won't be described here. The electronic schematic is shown on Fig. 4.12. The different electronics components and protocols used and their performances will be described. Namely, the I2C protocol, the multiplexer, the Arduino Mega 2560 and a miscellaneous electronics components such as a push button and a LCD screen.

4.3.2 The I2C protocol and the I2C multiplexer

The BNO055 are read using the I2C protocol, through an I2C multiplexer.

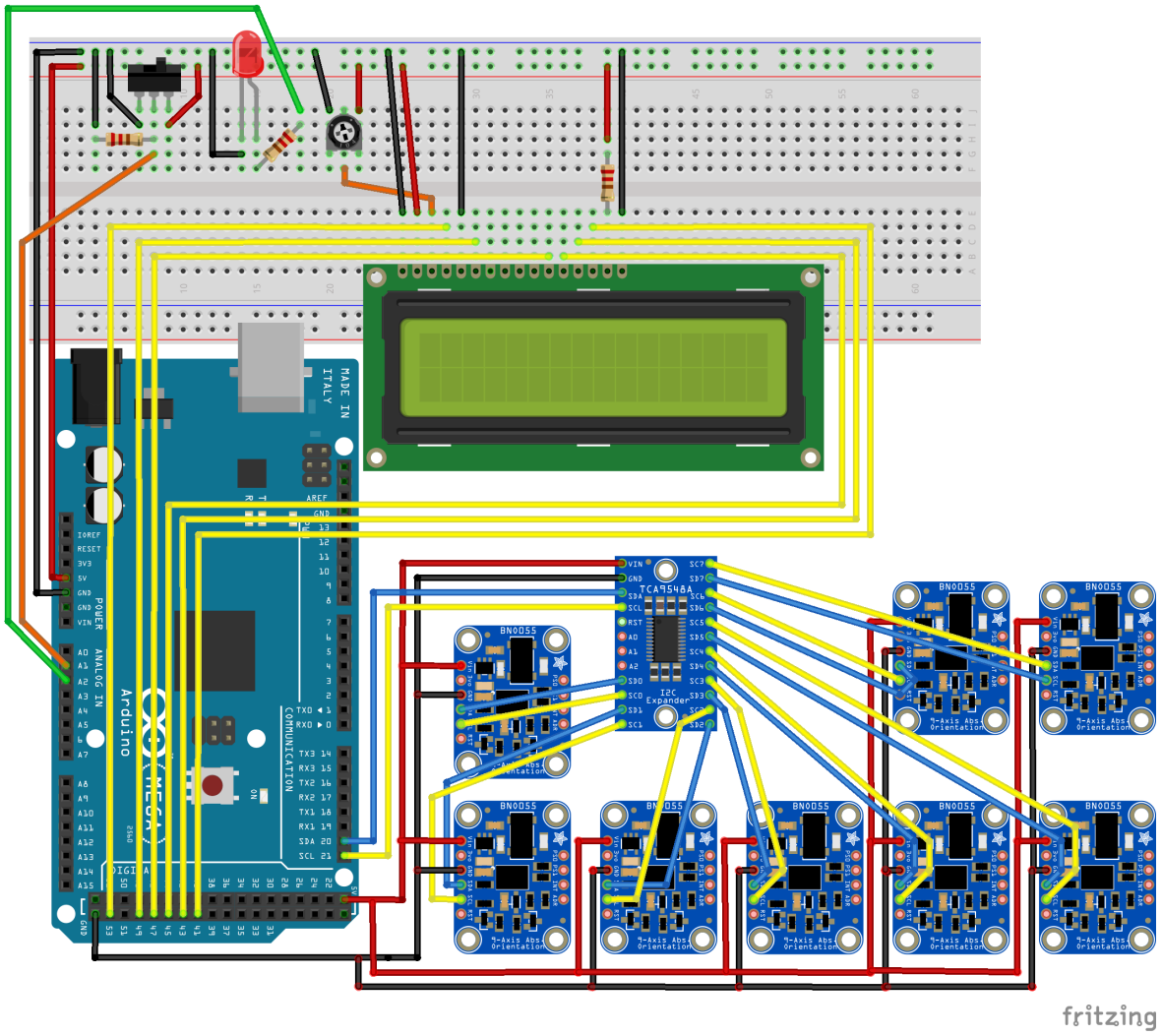


Figure 4.12: Experimentation electronics circuit wiring. The Board used is an Arduino Megaa 2560. The LCD screen displays useful information such as the calibration state or events such as a successful start or an unplugged IMU. The TCA9548A is used as a multiplexer to connect 8 BNO055 IMUs. A switch button is used to activate the calibration state streaming on the LCD screen. This function is used to check that the IMUs are well calibrated while streaming. A LED displays the state of the calibration streaming.

I2C is a communication protocol to communicate between masters and slaves. A master sends a message to a slave, and the slave sends back an message - or not, depending on the master's message and the device. I2C speed is 100kbps in standard mode, 400 kpbs in fast mode, 3.4 Mbps in high speed mode and 5 Mbps in Ultra fast

mode. Too high speed, such as too long or badly isolated wires, can lead to interference and then non-working communication. I2C is serial, because two transmission lines are used to send and receive data, and that data is continuously sent and received one bit at a time. I2C is synchronous, which means that the time taken to do the transfer will be unavailable for the program (the communication will not return control to the program until the whole data is sent or received). The I2C communication uses 4 lines, as shown on Fig. 4.13:

- VCC, typically 3v3 or 5v
- Serial Data (SDA)
- Serial Clock (SCL)
- Ground

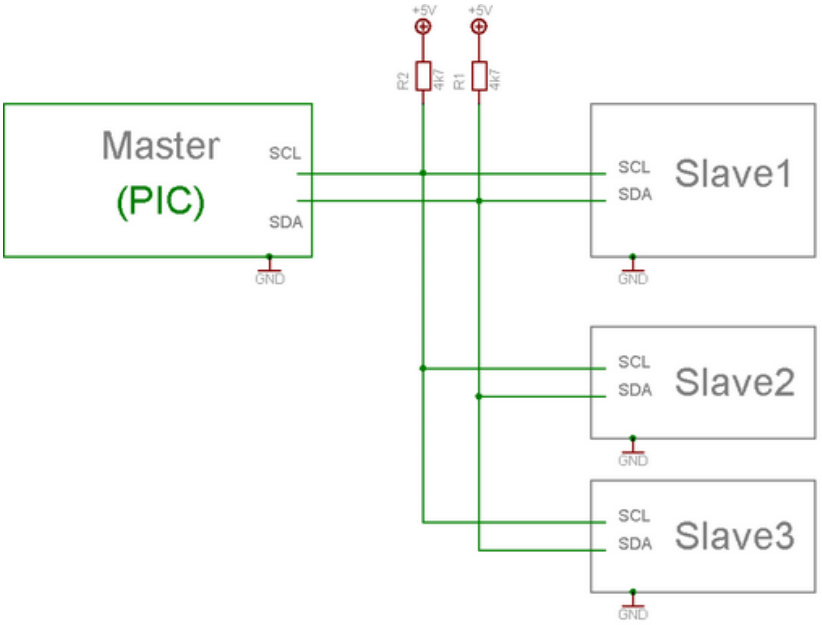


Figure 4.13: Wiring Master/Slave in the I2C protocol

Each slave has a unique address. For most of the cases, the device's address is built-in and cannot be changed, or changed to only one other address through an ADR pin, like for the BNO055. To use several devices with the same address using the same I2C line, a multiplexer is used, in our case a TCA9548A. The multiplexer has its own I2C address, and is connected to I2C slaves through individual channels. The master sends messages to the multiplexer to connect or disconnect such or such channel to the SDA and SCL lines. Then, when the master sends a message to another address, the multiplexer transmits it to the connected channels. See Fig. 4.14.

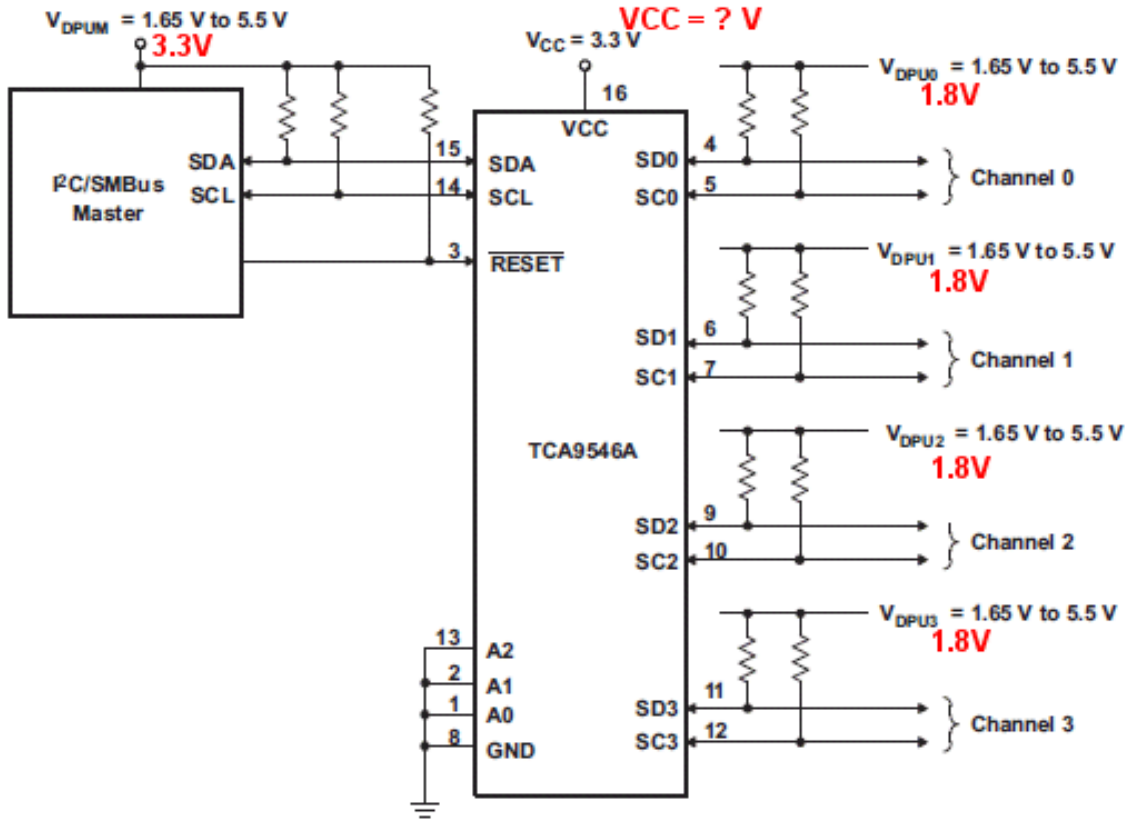


Figure 4.14: I2C multiplexer function circuit. The multiplexer receives a message from the master to connect or disconnect such or such channel to the SDA and SCL lines.

The I2C message structure is shown on Fig. 4.15. It begins by an address using 7 or 10 bits, depending on the I2C standard. In the BNO055 and the TCA9546A case, a 7 bits address frame is used.

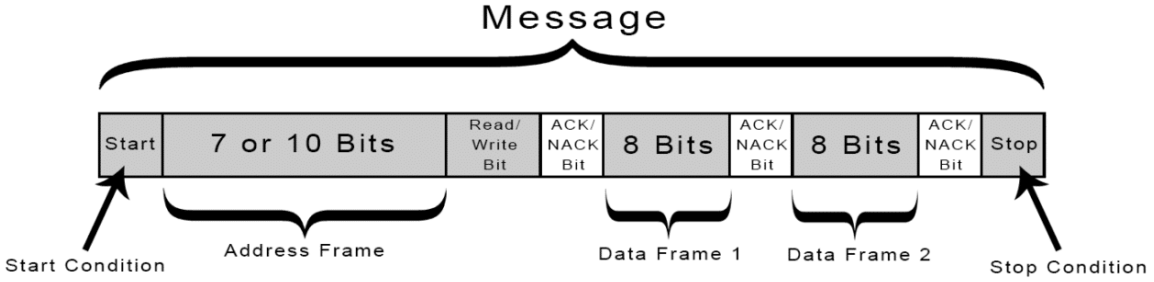


Figure 4.15: Message structure in I2C protocol

4.4 ROS package

In this part will be described the ROS package "sesa" (State Estimator for Soft Arm), with which the experiments were run.

First, a rough overview of the package will be given. Then, the nodes and topics will be listed, to complement the previously given overview. The launch process will be described. The launch node will be described in the same part. The workflows will be explained, for the calibration mode and for the streaming mode. The firmware package "sesa-firmware", a package used by sesa to push Arduino code into an Arduino Mega 2560 board, will be shown. Eventually, the nodes here will be described in detail one by one.

The code can be found in this GitHub repository.

Note: there are two packages in the repository, sesa and sesa-firmware. Later in the text, "the package" refers to the sesa package.

4.4.1 An overview

The sesa package has two distinct working modes.

1. **Stream mode**: the user streams BNO055 IMUs sensing under the form of quaternions in order to estimate the shape of a continuum soft arm, and displays it on an interface.
2. **Calibration mode**: ROS streams the calibration while the user calibrates its IMUs.

Let's describe more in depth the stream mode from the user perspective. The state is estimated using IMUs, using the PCC-extended rigid body method presented in the methods section. The number of IMUs used can vary between 1 and 8. Many parameters can be set to model the real soft arm, such as the orientation of the base of the arm, the placement of the IMUs along the arm, the number of links to approximate the PCC model between two IMUs, and many others. The state estimation is saved at specific points indicated by the user called markers. The measurements can be recorded into .bags files. Previously saved calibration can be pushed into the IMU when re-booting.

Using the calibration mode, the user can select the file in which the calibration will be saved.

On a pure hardware perspective, for both modes, the IMU sensing streaming is done through rosserial, using an Arduino Mega 2560. A multiplexer allows a communication through I2C. There are two different operating firmware for the Arduino

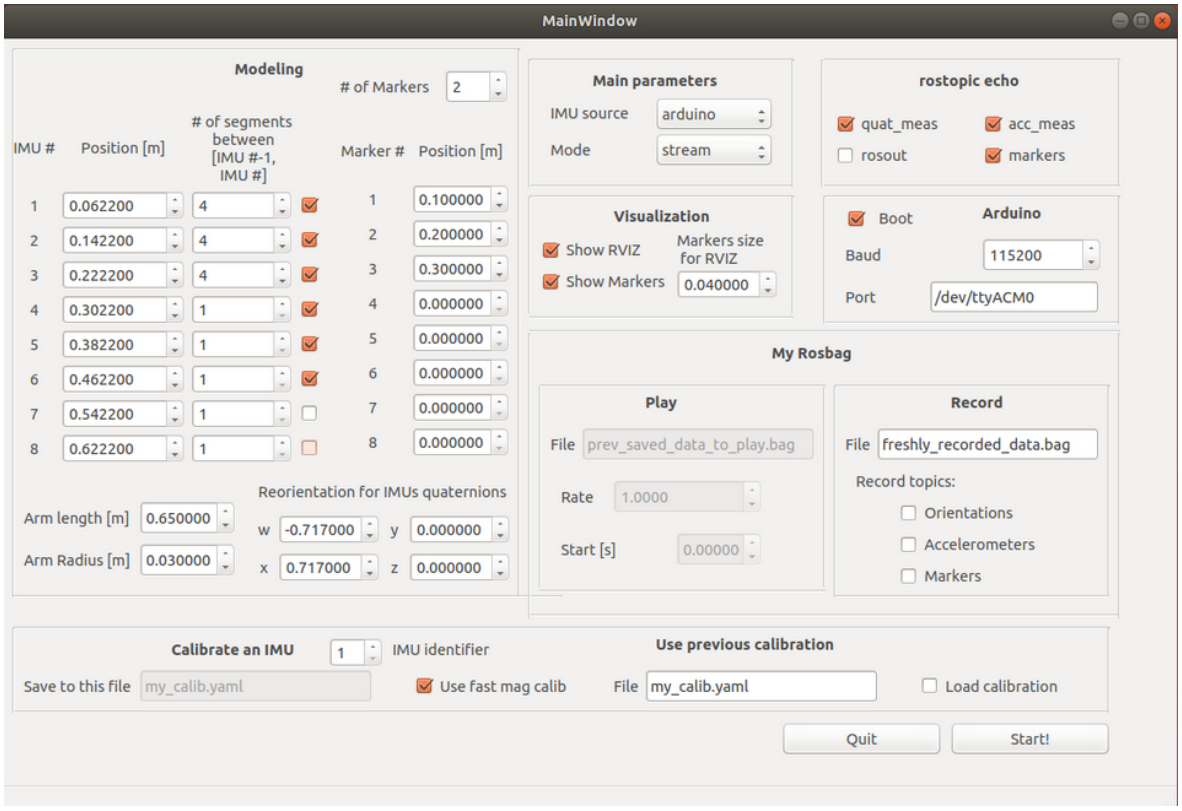


Figure 4.16: The Graphical User Interface (GUI) of the ROS package.

Mega 2560, one for the stream mode and one for the calibration mode. Another ROS package on the GitHub repository, the sesa-firmware package, can directly push code to the Arduino board when requested, and is called on request by the sesa package.

In the case of the stream mode, the source can be set to arduino or to my_rosbag. If the source is my_rosbag, measurements previously recorded into .bag files are replayed. This would allow for example to model the same arm several times with different models.

The goal of this work is to be used after this thesis, so the ROS package is equipped with a GUI (Fig. 4.16) and extensive documentation to make it easy to use for researchers interested in using this approach.

4.4.2 Nodes and topics

ROS is structured with nodes and topics. Nodes are publishing and subscribing to topics, and therefore communicate with other nodes. They can also run programs, hence the modeling performed in Estimator.

The estimator models the IMU using the quaternions measurements. The "Save Calibration" node saves the calibration state pushed to the parameters by the Arduino board. The "ROS Serial Arduino" node handles the Serial communication with the Arduino board. The visualization nodes (RVIZ, node_robot_state_publisher) display the arm state in real time through a visual interface. The "My Rosbag Record" node records the quaternions stream. The "My Rosbag Play" node plays the quaternions stream of a previously recorded experiment.

In this code, we use the following nodes:

- Estimator
- RVIZ
- node_robot_state_publisher
- My Rosbag
- Save calibration
- Arduino
- Launch (only used at the beginning of the program)

We use the following topics:

- /quat_meas
- /acc_meas
- /markers
- /JointState
- /calib_meas
- /calibration_status

4.4.3 Launch process and node

The program is launched by typing the following lines of code in a terminal. A config file, stored in the folder /config, can be passed as an argument. By default, my_config.yaml will be used.

```
1 $ cd [PATH TO YOUR CATKIN WORKSPACE]/catkin_ws
2 $ roslaunch sesa my_launch.launch config_file:="my_config.yaml"
```

The .launch ROS execution routine starts the launch node.

Right when the launch node starts, the GUI shown in Fig. 4.16 pops up. The user can edit the configuration using the GUI, such as selecting the stream or calibration mode, or modeling the arm with segments. The calibration is saved in a configuration file, in the repo /config. When the user is done with setting up his calibration and presses the start button, the GUI is closed and the launch node routine keeps going on.

Now that the configuration is known, the launch node builds the urdf file for the visualization using the make_my_urdf.py source file. It pushes the configuration file to rosparam, to make the configuration accessible to all of the nodes, including the one handled by the Arduino board. If the user checked the boot box on the GUI, the launch node calls the sesa-firmware package to push the firmware directly to the Arduino board.

Eventually, the launch node is starting all of the other nodes depending on the configuration file. The configuration determines which nodes are getting started. The nodes use the configuration as an input. Therefore, the code is modular and adapts to the need of the user. The configuration is saved as a configuration file, then pushed to rosparam. For example, the user can set the parameter source to arduino, and then the ROS Serial Arduino node is started. If source is set to rosbag, the My Rosbag Play node will be launched and will play the file /my_rosbag/to_play/file, with a rate /my_rosbag/to_play/rate and starting at time rosbag/to_play/start.

4.4.4 Workflow of the ROS package

This part will describe how the ROS package nodes communicate with each other for each mode, stream and calibration. The flowcharts in Fig. 4.17 and 4.18 help having a better understanding of this part.

For both modes, the package starts by launching the GUI and selecting a configuration. The configuration is saved into a config file. Then, when the user presses start, the Launch node, written in launch.py, is started and starts the other nodes. More details are presented in part 4.4.3.

Workflow of the calibration mode

This part is shown in Fig. 4.17.

The Arduino board reads the calibration of the BNO055 and publishes it into the topic /calib_meas. The calibration includes what the calibration constants are and how well does the BNO055 evaluates the accuracy of the calibration. The message type is the following:

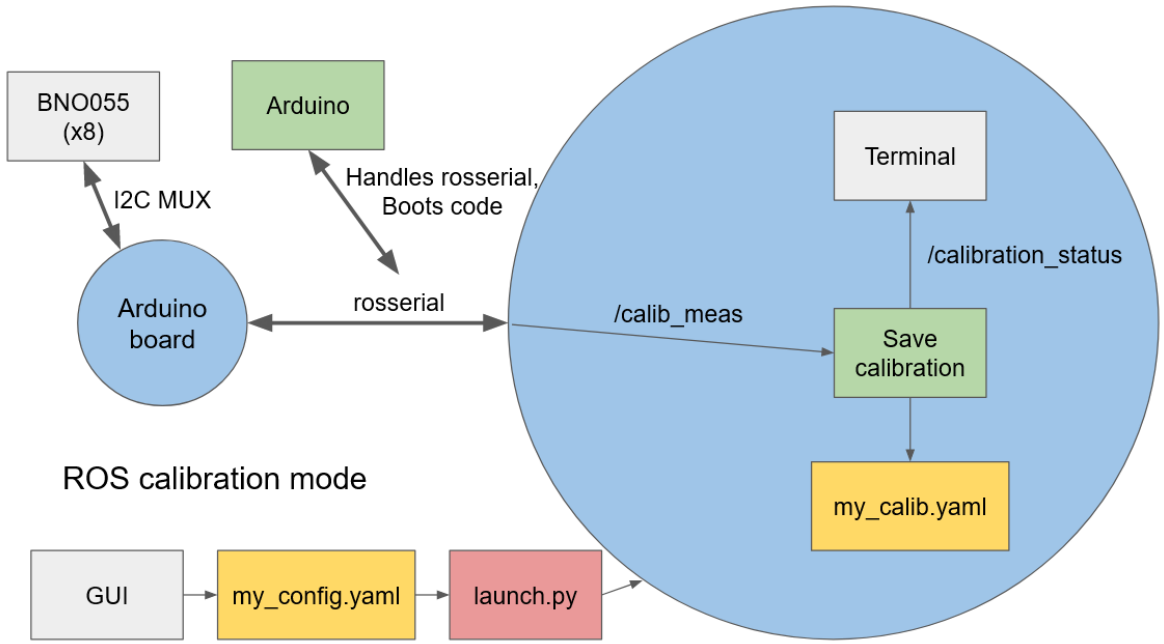


Figure 4.17: Flowchart of the ROS package in calibration mode

```
std_msgs/Header header
uint8 ID
uint8 sys
uint8 acc
uint8 gyro
uint8 mag
int16 off_accX
int16 off_accY
int16 off_accZ
int16 off_magX
int16 off_magY
int16 off_magZ
int16 off_gyroX
int16 off_gyroY
int16 off_gyroZ
int16 rad_acc
int16 rad_mag
```

The connection between the Arduino board and the computer is done through a serial connection, handled by the Arduino node, based on the rosserial package. The node Save Calibration is saving the calibrations into the selected calibration file, here my_calib.yaml. It also publishes the calibration status into the /calibration_status, i.e

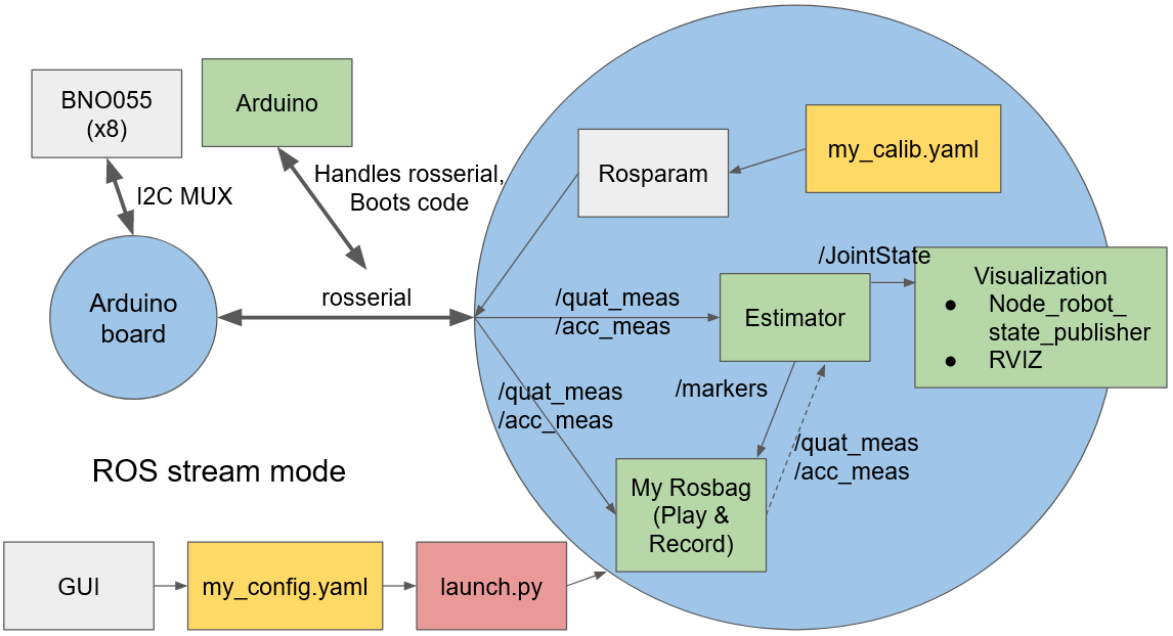


Figure 4.18: Flowchart of the ROS package in stream mode

how accurate is the calibration, through an estimation of the overall system (sys), the accelerometer (acc), the gyroscope (gyro) and the magnetometer (mag). The message type is the following. The string head is"sys acc gyro mag". The string imu_i shows the sys, acc, gyro, and mag calibration for the imu #i.

```
std_msgs/Header header
string head
string imu_1
string imu_2
string imu_3
string imu_4
string imu_5
string imu_6
string imu_7
string imu_8
```

Workflow of the stream mode

This part is shown in Fig. 4.18.

The source corresponds to where the quaternions come from. They can come either from the Arduino board, from a rosbag using the node My Rosbag Play, or from an external publisher. We will assume here that the source is Arduino. If the source is My

Rosbag Play, the workflow is roughly the same except that there's no serial connection with Arduino and that the data comes from a previously recorded data bag.

When the system is launched, if the user decided to push a previously saved calibration into the IMUs (parameter calib/use_previous is true), my_calib.yaml is pushed to rosparam. Rosparam is a ROS feature storing global parameters, that can be get and set by all nodes. Then, the Arduino board will get these params from rosparam and push them onto the IMUs as calibrations.

Then, the Arduino board reads the IMUs through an I2C multiplexer, and publishes these measurements on the topics /quat_meas and /acc_meas. The message types are the following, resp. for acc.msg and quat.msg:

```
std_msgs/Header header
uint8 ID
float32 x
float32 y
float32 z
```

```
std_msgs/Header header
uint8 ID
float32 w
float32 x
float32 y
float32 z
```

The serial connection is ensured by the Arduino node, based on the rosserial node.

The topics /quat_meas and /acc_meas are read by two nodes, Estimation and My Rosbag. My Rosbag reads and records the data. Estimation will compute the model given the measurements, and publish the positions and orientations of the bodies to JointState. This topic will be read by the node_robot_state_publisher node, which will transfer it to RVIZ, the display module.

4.4.5 Sesa-firmware package

This package compiles the Arduino code using the Aruidno IDE kernel and loads it to an Arduino Mega 2560. The package use the generate_arduino_firmware() function from the rosserial_arduino package. We followed a tutorial that can be found here.

It should be mentioned that, using Ubuntu 18.04 and ROS Melodic, the compiler should be C++11, otherwise there are compatibility issues. This is set in the CMakeLists.txt of the package.

Writing this package required to generate custom message types for Arduino. A tutorial to do this can be found here.

This project has 2 possible firmwares, one per mode:

- **stream**, the plug and play firmware. This firmware streams the IMU data. If the parameter calib/use_saved is true, the firmware loads a pre-recorded calibration calib/saved_file saved in calib/saved_path.
- **calibrate**. This firmware streams the calibration state. This ROS package can save the stream calibration data into a local file through the save_calibration.py script.

For both modes, the IMU is set in NDOF mode, which auto-calibrate across time using the embedded firmware of BNO055 using Fast Magnetometer calibration.

4.4.6 Estimator node

About

This node is started if mode is stream. It streams the quaternions from /quat_meas. If rviz_gui is true, estimator publishes to /robot_state_publisher the state of the robot, showing the markers positions with green balls. If topic/markers is true, estimator publishes the markers positions on /markers. The node is written in /scripts/sesa/estimator.py.

Configuration parameters used

- imus
  - amount
  - enabled
  - offset_quaternion
  - positions
- markers
  - amount
  - enabled
  - positions
  - radius
  - show

use

- segments
- rviz_gui

4.4.7 Save calibration node

About

This node is started if mode is calibrate. It streams the calibration state pushed in the parameters by the arduino board, and saves it to the file calib/calibrate_file. The node is written in /scripts/sesa/save_calibration.py.

Parameters used

- calib

  calibrate_file

  id

  use_fast_mag

4.4.8 ROS Serial Arduino node

About

This node is started if source is arduino. If mode is calibration, source is automatically set to arduino. This node handles the Serial communication with the Arduino board. This node is using the serial_node.py script of the public rosserial_arduino package. **If the user wants to change the baud rate, he needs to change it in the firmware too!**

Configuration parameters used

- Arduino

  baud

  port

4.4.9 Visualization (RVIZ) nodes

About

This block contains two nodes:

- node_robot_state_publisher

- rviz

These nodes are started if rviz_gui is true and if mode is stream. The ROS package rviz is launched and displays the arm state in real time through a visual interface. It uses the robot description /robot_description created under the urdf format using the source file /src/make_my_urdf.py during the launch. The positions and orientations of the arm are read on the /JointState topic by node_robot_state_publisher and transferred to the urdf node for visualization.

Parameters used

- robot_description

4.4.10 My Rosbag Play node

About

This node is started if source is rosbag and mode is stream. It is based on the ROS Package rosbag and it plays the quaternions stream of a previously recorded experiment into the /quat_meas topic from the file /my_rosbag/to_play/file.

Configuration parameters used

- my_rosbag/ to_play

  file

  rate

  start

4.4.11 My Rosbag Record node

About

This node is started if source is arduino and mode is stream. It is based on the ROS Package rosbag and it records the quaternions stream of the /quat_meas topic into a file /my_rosbag/to_record/file.

Configuration parameters used

- my_rosbag/ to_record

  file

  accelerometers (bool)

  markers (bool)

quaternions (bool)

4.5 Validation experiments: comparing our method to motion capture

In this section, we will present how we designed our experiments, first overall, and then each model independently. The description includes the number of IMUs used, the use of motion capture as ground truth, and the error metrics definitions. Eventually, details about the experimentation implementation will be given.

4.5.1 Outline of the validation experiments

Two models are presented in Section 3. To evaluate the accuracy of our models, motion capture is used as ground truth. While the arm is moved, measurements from the IMUs and the motion capture system are recorded. The IMU data is then used to construct models according to Section 3 offline. To evaluate how model performance is affected by the density of IMUs along the arm, both models are constructed using the data from two, four, and eight IMUs.

In all experiments, error is defined as the Cartesian distance between the estimated positions of the motion capture markers and their actual positions. The positions of the motion capture markers are estimated from the models and compared to the actual positions recorded by the motion capture system. The error is evaluated on two points along the arm, Marker 1 and Marker 2, respectively at distances of 360 mm and 600mm from the base along the arm. At each of these points, two error metrics are defined.

Absolute errors e_1 and e_2

e_1 and e_2 are respectively the Cartesian distances between the Motion Capture Marker 1 and 2, and their position estimations. These metrics are named the absolute errors.

Relative errors E_1 and E_2

E_1 and E_2 are respectively e_1 and e_2 divided by their distances to the arm's base along the spine l_1 and l_2. These metrics are named the relative errors.

They are shown in Fig. 4.19.

4.5.2 Rigid-body model experiment

The rigid-body model is implemented using two, four, and eight IMUs (and thus with an equivalent number of links). The implementation is described in part 4.4.11. The positions of the IMUs along the arm are shown in Table 2.

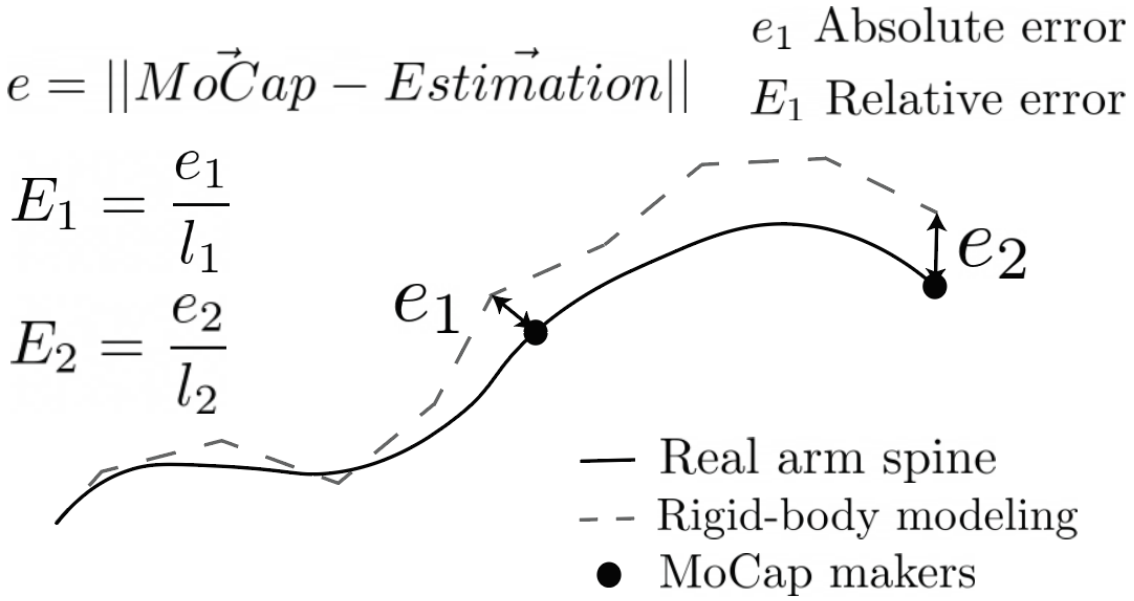


Figure 4.19: Error measurement metric to compare the motion capture data to the estimation. On this figure, the estimation is a Rigid-body model using 8 links.

We set $L_0 = 0$mm since the first link of length L_0 (see Fig 3.1) empirically induces a vertical error offset. When using two and four IMUs, the length of the arm can be divided into two and four equal sections, respectively, with an IMU mounted to the midpoint of each section. For eight IMUs, when the length of the arm is divided into 8 equal sections, the segmentation is made such that the IMUs lie at the endpoints of the sections instead of the center due to their physical spacing.

4.5.3 PCC-extended Rigid-body model experiment

The PCC-extended Rigid-body model is tested w.r.t. two variables: the number of links, and the number of IMUs.

First, we fixed the number of IMUs to the maximum, i.e. eight IMUs, and we used 8, 16, 32, and 64 links. Results shown in Fig. 5.4 suggests that 16 links are sufficient to approximate the maximum curvature exhibited by our arm.

Then, to explore the impact of the number of IMUs on pose estimation accuracy, we generated a 16 links model using data from two, four, and eight IMUs. The position of the IMUs used along the arm are shown in Table 2.

IMU #	Base*	1	2	3	4	5	6	7	8
pos [mm]	0	80	160	240	320	400	480	560	640
2 IMUs	E		R		E		R		E
4 IMUs	E		X		X		X		X
8 IMUs	E	X	X	X	X	X	X	X	X

Table 2: This table shows which IMUs are selected for a given number of IMUs used to feed each model. Pos corresponds to the position of the IMU along the arm. "R" stands for "rigid-body only", "E" stands for "PCC-extended rigid-body model only", and "X" stands for both.
*base is the orientation of the arm's base, not counted as an IMU but used in the PCC-extended rigid-body model.

4.5.4 Timing synchronization

To ensure the same sampling frequency on both the IMU model and motion capture signals, linear interpolation across time is processed on the model output. The time delay between both signals is computed using cross-correlation and removed.

4.5.5 Conditions of the data acquisition

In a motion capture environment, the arm is fixed, base up, on an elevating structure (See Fig. 2.6). Over a 250sec trial, the arm is moved manually with a stick attached to the arm between the two motion capture markers. During the trial 8,000 samples are collected, The median of the arm's moving speed is 0.1ms^{-1} at marker 1 and 0.2ms^{-1} at marker 2, reaching speeds up to 0.6ms^{-1} (Fig. 4.20).

The arm is moved in different modes such as oscillating (going back and forth on each side of the structure, the tip reaching positions below the base w.r.t the z-axis), buckling (pseudo vertical position and contractions along the z-axis), and "dog chasing its tail" (the arm making circles around the z-axis while its shape forms a question mark).

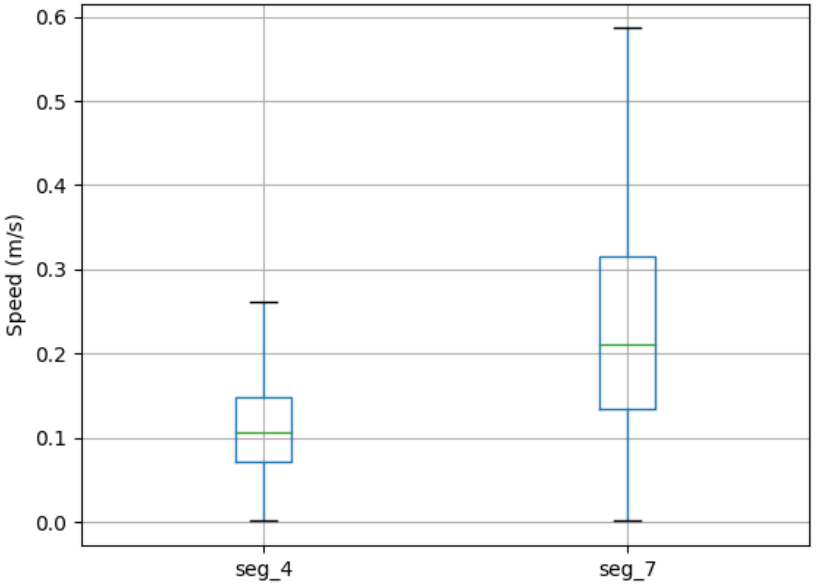


Figure 4.20: Speed distribution at the two motion capture markers positions.

5 Results

In this section will be discussed the results of the experimentations. First, the raw performance of the two models using 8 IMUs will be exposed, using absolute and relative error. The impact of the IMU density will be studied for both models, as well as the impact of the number of segments for the PCC-extended model. Then, computational considerations will be given. Eventually, an estimation of the sources of errors will be done.

5.1 Absolute and relative error for 8 IMUs

In this part will be presented the absolute and relative error for both models using 8 IMUs. Overall, the PCC-extended Rigid-body model outperforms the Rigid-body model by approximately 25%, reaching a median relative error $\leq$10%. This corresponds to a median error of 36.7mm at Marker 1 (360mm) and 54.6mm at Marker 2 (600mm).

5.1.1 Absolute error

On Fig. 5.1, the state estimation for the PCC-extended Rigid-body model and the Rigid-body model are represented on the RVIZ interface of the ROS package. In this specific configuration, the arm's tip touches its base. This figure gives us an intuition of the performance difference between both models. We can see that the PCC-extended Rigid-body model performs better.

The table 3 gives us specific numbers regarding the median and 3rd quartile of the error of both models w.r.t the motion capture system. These numbers are shown under the box plot form on Fig. 5.3a and Fig. 5.3b.

As expected, the absolute error is higher for the Marker 2 than for Marker 1. The median error for the Rigid Body model is 47.9mm at Marker 1 (360mm) and 70.8mm at Marker 2 (600mm). The median error for the PCC-extended Rigid Body model is 36.7mm at Marker 1 (360mm) and 54.6mm at Marker 2 (600mm).

The 3rd quartile for the Rigid Body model is 63.7mm at Marker 1 (360mm) and 97.5mm at Marker 2 (600mm). The 3rd quartile for the PCC-extended Rigid Body model is 48.6mm at Marker 1 (360mm) and 82.8mm at Marker 2 (600mm).

5.1.2 Relative error

The table 4 shows the relative error for both models.

The median error for the Rigid Body model is 13.3% at Marker 1 (360mm) and 11.8% at Marker 2 (600mm). The median error for the PCC-extended Rigid Body

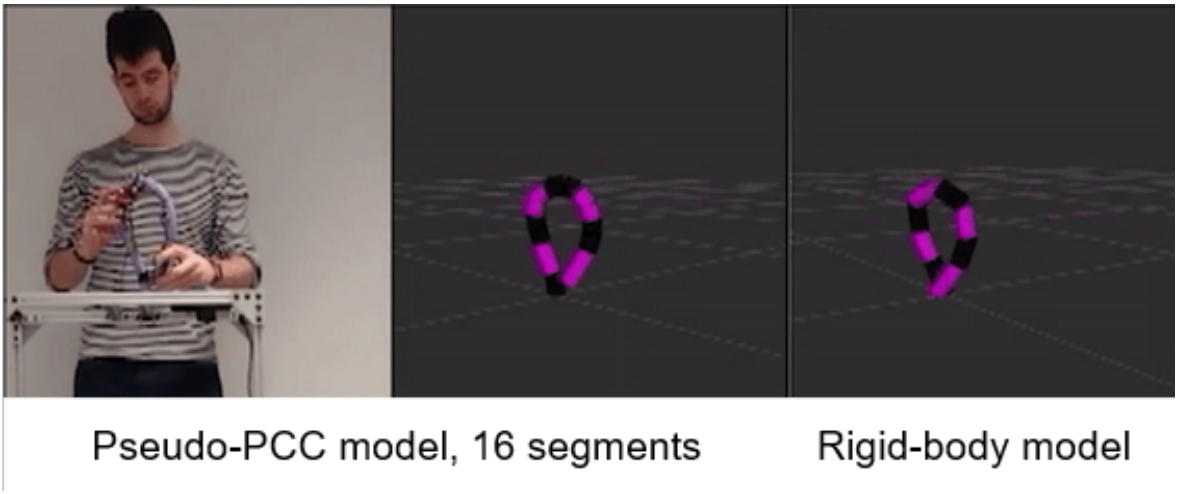


Figure 5.1: Model estimation of the real-world soft arm (left figure) using the PCC extended Rigid-body model (middle figure) and the Rigid-body model (right figure). In this specific configuration, the arm's tip touches its base.

Marker	Quartile	Rigid-body	PCC-extended
M1 (L=360mm)	Median	47.9mm	36.7mm
M1 (L=360mm)	Q_3	63.7mm	48.6mm
M2 (L=600mm)	Median	70.8mm	54.6mm
M2 (L=600mm)	Q_3	97.5mm	82.8mm

Table 3: Median and the third quartile (Q3) of the absolute error for the rigid body model and the PCC-extended Rigid-body model using eight IMUs.

model is 10.2% at Marker 1 (360mm) and 9.1% at Marker 2 (600mm).

The 3rd quartile for the Rigid Body model is 17.7% at Marker 1 (360mm) and 16.25% at Marker 2 (600mm). The 3rd quartile for the PCC-extended Rigid Body model is 13.5% at Marker 1 (360mm) and 13.8% at Marker 2 (600mm).

Marker	Quartile	Rigid-body	PCC-extended
M1 (L=360mm)	Median	13.3%	10.2%
M1 (L=360mm)	Q_3	17.7%	13.5%
M2 (L=600mm)	Median	11.8%	9.1%
M2 (L=600mm)	Q_3	16.25%	13.8%

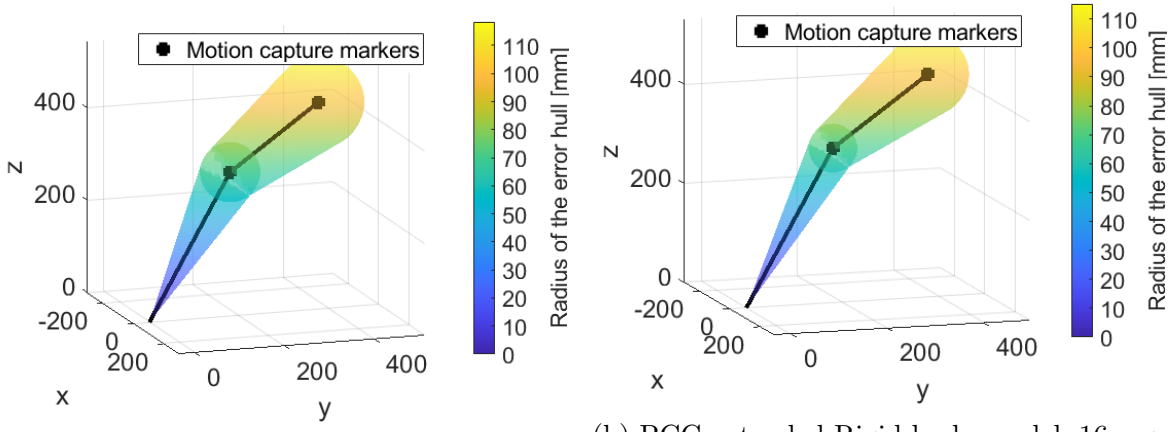
Table 4: Median and the third quartile (Q3) of the absolute error for the rigid body model and the PCC-extended Rigid-body model using eight IMUs.

Two different trends should be remarked:

- Whereas the absolute error is higher for the Marker 2 than for Marker 1, the median relative error tends to decrease along the arm.
- The standard deviation tends to increase along the arm. The Q3 is slightly decreasing along the arm for the Rigid-body model, and slightly increasing for PCC-Extended Rigid-body model.

The Q3 part of the table 4 is represented under the hull form on Fig. 5.2a and Fig. 5.2b.

For the Rigid-body model (Fig. 5.2a), hull width w.r.t the spine length at Markers 1 and 2 are respectively 17.7% and 16.25% . For the PCC-extended Rigid-body model (Fig. 5.2b), hull width w.r.t the spine length at Markers 1 and 2 are respectively 13.5% and 13.8%.



(a) Rigid-body model, 8 segments

(b) PCC-extended Rigid-body model, 16 segments

Figure 5.2: Representation of the Q3 error hull, containing 75% of the state estimations. The black line in the middle corresponds to the ground truth. The black points corresponds to the motion capture markers.

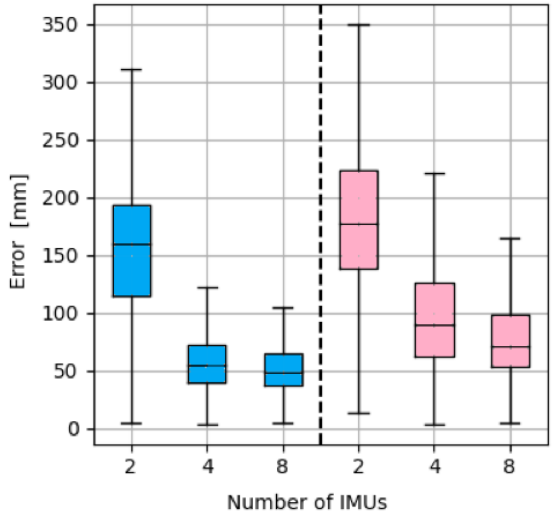
5.2 Reducing the amount of IMUs

The IMU density is a critical parameter for the estimation accuracy. The goal of this part is to study the influence of the IMU density over the accuracy, depending on the model. Therefore, we gain visibility on the amount of IMUs required to obtain a decent accuracy.

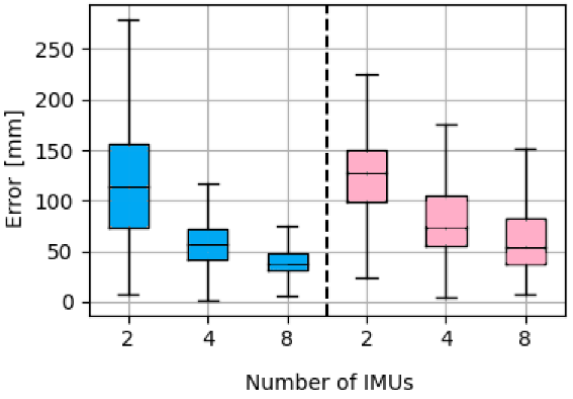
The way IMUs were outfitting the arm for 2, 4 and 8 IMUs is shown on Tab. 2. The results of the model estimation w.r.t the number of IMUs is shown on Fig. 5.3a for the Rigid-body model, 2, 4, and 8 segments, and Fig. 5.3b, for the PCC-extended Rigid-body model, 16 segments.

As expected, the accuracy increases with the density of IMUs. The PCC-extended Rigid-body model presents exactly what we could expect intuitively, i.e a regular improvement of the accuracy as the IMU density increases.

The Rigid-body model present interesting results. Using 4 IMUs, the performance of the Rigid-body model is comparable to the PCC-extended model, whereas using 8 IMUs, the performances are significantly poorer. These results tends to indicate a limitation of the Rigid-body model above a given threshold of IMU density that the PCC-extended Rigid-body model doesn't have.



(a) Rigid-body model, 2, 4, and 8 segments



(b) PCC-extended Rigid-body model, 16 segments

Figure 5.3: Error between the models estimations and the motion capture measurement, depending in function of the number of IMUs. The blue box-plots corresponds to the Marker 1. The pink box-plots corresponds to the Marker 2.

5.3 Increasing the amount of segments of the PCC-extended Rigid-body model

This part is an application of the theory developed in part 3.2.3, i.e the approximation of the PCC-extended Rigid-body model using fixed-sized cylindrical segments.

On the figure Fig. 5.4, we can see the that there's a significant improvement between using 8 segments (i.e the Rigid-Body model) and using 16 segments (basically, chopping each segment into 2). Though, a plateau is reached at 16 segments, and increasing further the number of segments doesn't improve the accuracy.

These results corroborate the model presented in Fig. 3.8, i.e the approximation error decreases proportionally to $\frac{1}{(N_{segs})^2}$. If the accuracy doesn't improve over 16 segments, it means that the PCC model is approximated well enough by the PCC-extended Rigid-body model to perform as well as the PCC model itself. Also, the PCC model is not perfectly fitting the arm's behavior, which explains why the approximation doesn't need an extremely high accuracy for the model to reach its maximum performances.

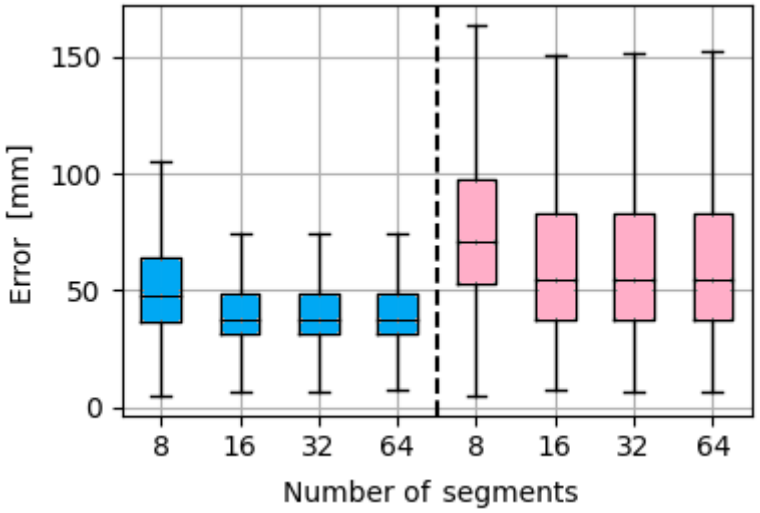


Figure 5.4: The error of the PCC-extended Rigid-body model using 8 IMUS and 8, 16, 32 and 64 segments. The blue box-plots corresponds to the Marker 1. The pink box-plots corresponds to the Marker 2.

5.4 Computational considerations

Both models can be computed in real time for two, four, and eight IMUs using a personal computer with an Intel® Core™ i7-10510U Processor (4.90 GHz) and 16GB of RAM.

A non-optimal implementation of the PCC-extended Rigid-body model could lead to not be constructed in real-time on the same machine, due to the added computational burden of repeatedly performing spherical linear interpolation.

5.5 Estimation of the sources of error

Our results shows, for the PCC-extended Rigid-body model, median errors of 36.7mm at Marker 1 and 54.6mm at Marker 2. We want to get to know better where this error comes from. By getting through the state estimation pipeline, we make the following list of error sources, and quantify the median error for each of them.

- IMU fixation on the arm.
- Sensor stacked inaccuracies.
- Latency of the sensor, inducing a delay and therefore an error.
- PCC-model inaccuracy w.r.t the soft arm behavior.
- PCC-model approximation using fixed-length segments.
- Motion capture markers placement.
- Motion capture image processing.

In this section, we will quantify these error to build the error estimation model shown on Fig. 5.11.

5.5.1 Sensors stacked inaccuracies

In this part, we will estimate the amount of error due to the sensor noise. In order to do so, we will model the noise using a Gaussian distribution, and then simulate the propagation of the noise along 1 and 8 segments. Then, we will analyze the results to get a scale of the sensor noise error along the arm.

Based on the BNO055 accuracy measurement shown in Tab. 1, we decided to model the sensor noise with a Gaussian noise $\mathcal{N}(\mu, \sigma^2)$ with $\mu = 0$ and $\sigma = 1$. This distribution is shown on Fig. 5.5.

Then, using this distribution, we generate random arms to understand how the noise impacts the overall accuracy. To do so, we create one arm, the original one, and we create noised arms by adding noise at each segment. The arms are created following the Rigid-body model, i.e a series of rigid segments of 80mm. Therefore, we model how the noise spreads through a real chain of IMUs.

On Fig. 5.7b, we use a 1 segment arm. On Fig. 5.7a, we use a 8 segment arm. At first sight, these figures seems to show that the error stacking of the sensors tends to increase the overall error (absolute error) as the number of segment increases, but that since some errors compensate previous errors, the error w.r.t the amount of segment (relative error) decreases.

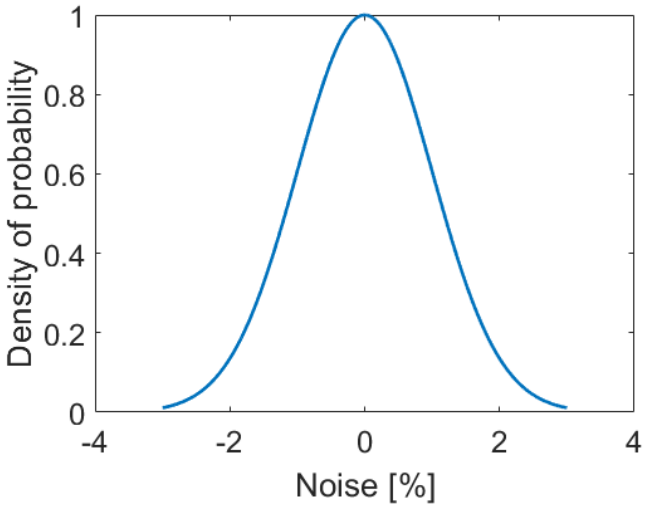
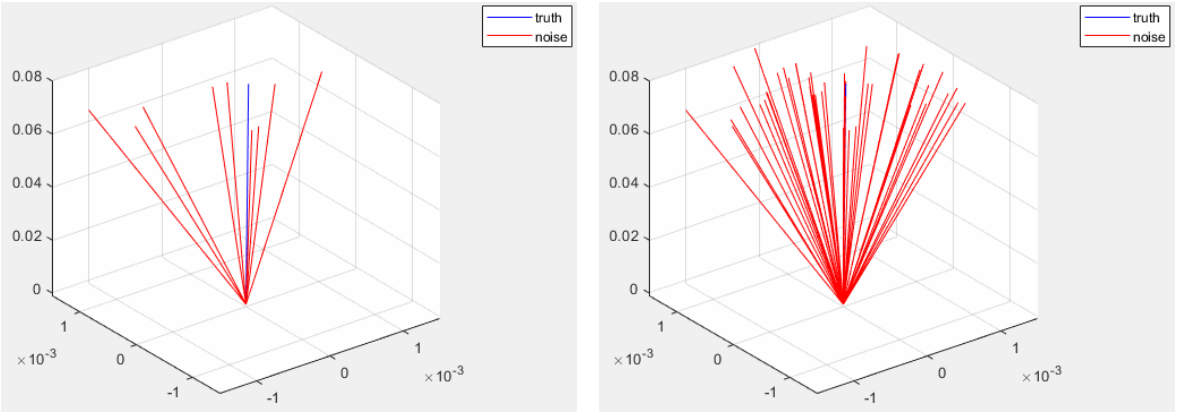


Figure 5.5: bno055 noise distribution



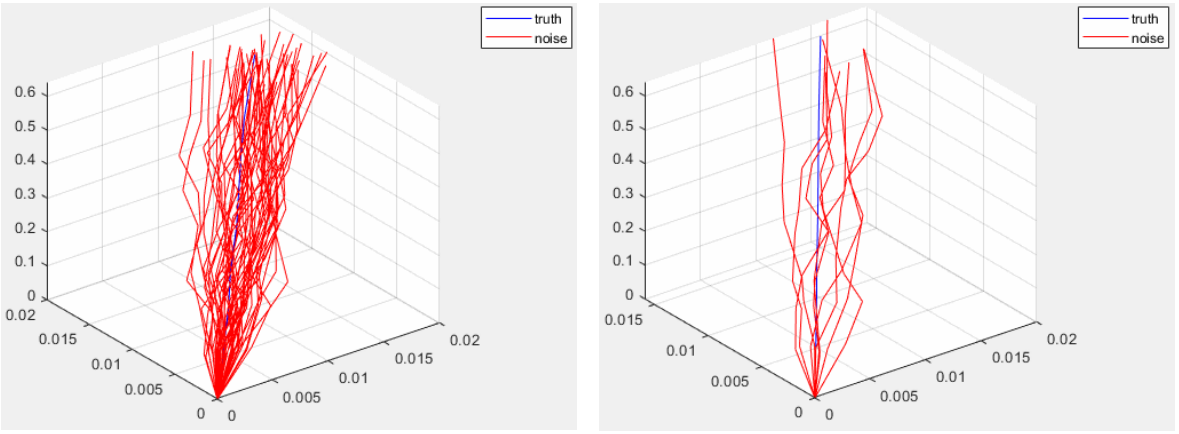
(a) 9 randomly generated arms (b) 50 randomly generated arms

Figure 5.6: Generation of noised 1-segment arms by adding random noise at each segment. The original arm is in blue. The noised arm are in red.

This first sight is confirmed by Fig. 5.9. On this figure, we can clearly see that the mean absolute error is increasing, as the mean relative error is decreasing.

From this experience, we can conclude several things:

- an estimation of the error w.r.t the segment number is on Fig. 5.9. The absolute error is increasing, such that at the end effector - the worst case scenario - it is below 3mm.
- The relative error is decreasing along the arm, which means that increasing the



(a) 50 randomly generated arms

(b) 9 randomly generated arms

Figure 5.7: Generation of noised 8-segment arms by adding random noise at each segment. The original arm is in blue. The noised arm are in red.

IMU density decreases the overall IMU noise. Indeed, if the IMU density is high, the noise compensates and therefore decreases overall.

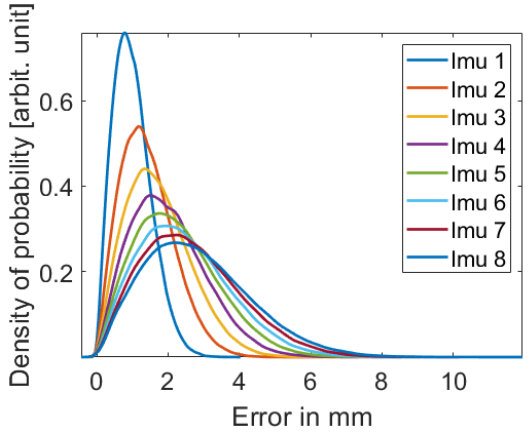


Figure 5.8: Density of probability function of the error for each IMU (each segment), based on simulation data.

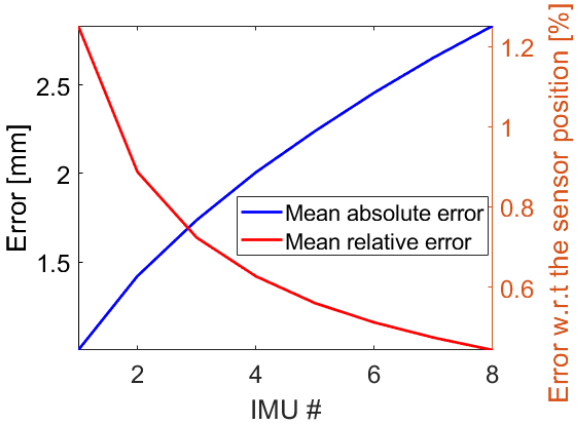


Figure 5.9: Mean of the arm error at each IMU (each segment), in absolute error and in relative error.

5.5.2 IMUs fixation on the arm

To quantify the noise induced by the angle difference between the IMU chip and the arm orientation at the IMU spot, we used an analog approach error to the stacked IMU error. We selected an arbitrary value of 1% noise as a median error. Since the

median of a Gaussian is 1.25σ, the model for the error is $\mathcal{N}(\mu, \sigma^2)$ with $\mu = 0$ and $\sigma = 0.8$. Using the stacked error model used for the previous section, the median error at Marker 1 is 1.5mm and at Marker 2 is 2.5mm.

The IMU fixation is shown on Fig. 5.10.

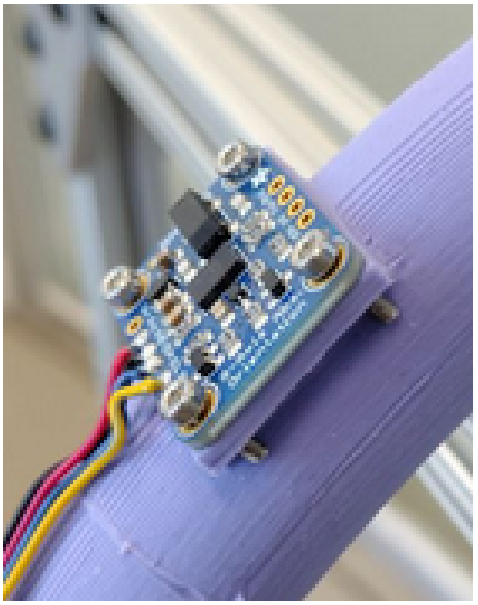


Figure 5.10: Fixation of the IMU on the soft arm

5.5.3 Latency of the sensor

The speed distribution at the 2 markers spots are shown on Fig. 4.20. Using the median speed, we estimate the latency error in position at each marker. The median speed for Marker 1 is 0.1m/s, and 0.21m/s for Marker 2.

- Marker 1: 0.1m/s * 20ms = 2 mm
- Marker 2: 0.21m/s *20ms = 4.2mm

5.5.4 Motion capture markers placement

Motion capture markers are place in groups, as shown on Fig. 4.5. This reduces the motion capture marker error by averaging them to find the centroid. To put a upper bound to this limit, we will estimate the error due to the motion capture markers placement to 3mm.

5.5.5 Motion capture image processing

According to the motion capture setup of the lab datasheet, the accuracy of the system is 1mm.

5.5.6 PCC model inaccuracy

We have no way to measure the PCC model inaccuracy properly. Though, we can quantify all of the other errors, and we know that the median error are 36.7mm for Marker 1 and 54.6mm for Marker 2. Therefore, we can deduce errors for the PCC model and its approximation using fixed-length segments. Using this method, the median error estimations are 27.1mm for Marker 1 and 41.2mm for Marker 2.

5.5.7 PCC model approximation using fixed-length segments

Part 3.2.3 shows that, using 16 segments to model the 8 IMUs arm, and having a maximum angle of $\frac{\pi}{2}$ between two IMUs, the maximum approximation error is 4%. Given the PCC model median error approximation, the maximum error coming from the PCC model approximation for a median error value are 1mm for Marker 1 and 1.5mm for Marker 2.

5.5.8 Overall estimation

This final plot (Fig. 5.11 shows us that the greatest source of error comes from the fit of the soft arm to the PCC model. The motion capture error is an error that is inherent to the validation experimentation, and therefore shouldn't be considered to evaluate our solution accuracy. It should be noted that the pure hardware error (the IMU fixation and the stacked IMU error) are quite low, even though we used cheap IMUs.

By looking at these plots, we are very optimistic that we could reach a high accuracy by increasing the IMU density. The PCC error decreases as the distance between IMUs is reduced. Also, we have shown that the IMU fixation and the stacked IMU error decrease as the density of IMU increases. Future work could explore this option.

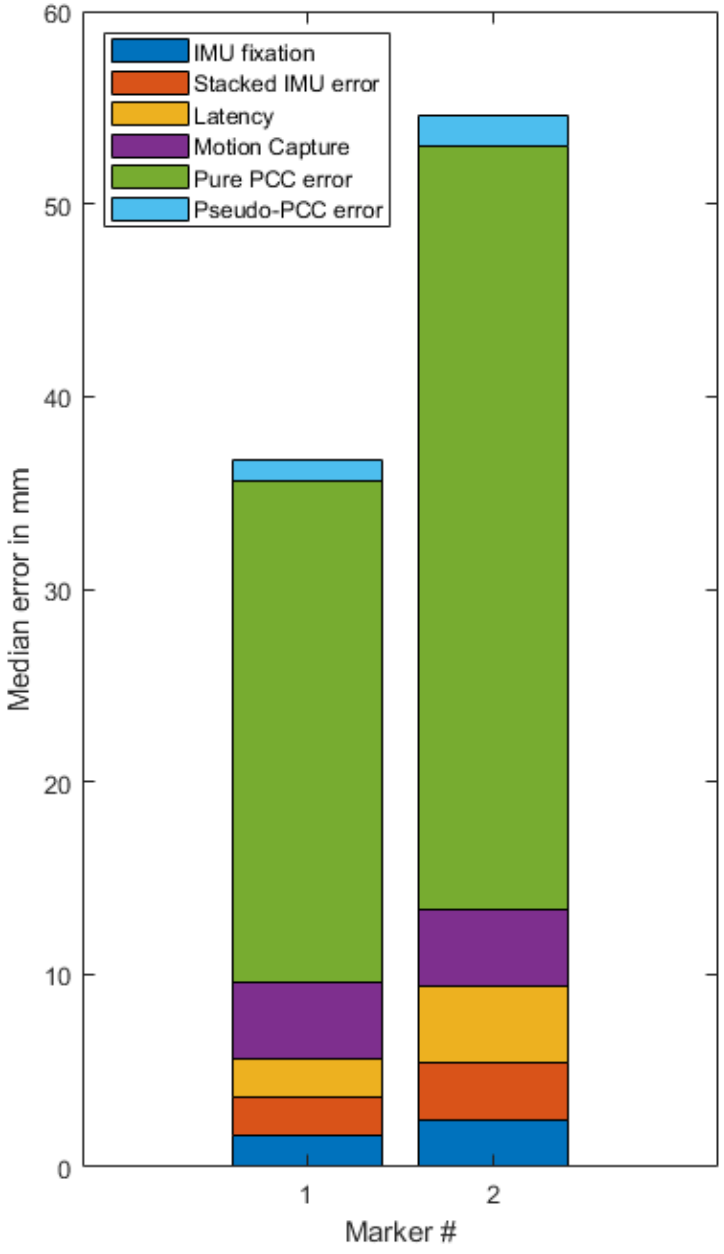


Figure 5.11: Estimation of the sources of error.

6 Discussion

6.1 Performances compared to the existing technologies

Our method showed a scale of 10% median error along the arm, with a decreasing error trend when increasing the arm's length. This performance is comparable to existing strain sensors for implementations such as a 15cm long finger. Motion capture is obviously performing better, but is expensive and must be used indoor in an occlusion-free environment.

Compared to strain sensors, the fact that our system relies on absolute orientation sensors allows us recover larger scale shapes. Indeed, when estimating the shape from strain sensors, the sensors are chained serially from one to another, and therefore the orientation error stacks up, increasing the error in position estimation. This drastically increases the error in position at a large scale. Using our method, the orientation error doesn't stack up since we use absolute orientation sensors. Only the position error stacks up, and therefore, at one scale lower than strain sensors.

Another advantage of our method is how inexpensive and available off the shelf it is. The BNO055 itself costs around 2$. Our method could therefore be used for a large public, at decent prices. The measurement are linear, and IMUs are available off the shelf. This makes this sensor easy to read and implement, and to replace in case of failure. Eventually, this sensor is mechanically easier to implement on various types of soft sensors.

Limitations of this sensor are obviously the IMU drift, and sensitivity to magnetic field variations. The BNO055 has an auto-calibration mode to compensate the IMU drift using a magnetometer, hence it get sensitive to perturbations in the magnetic field. Another argument that could be made is that IMUs can be relatively small (5.2 x 3.8 x 1.1mm for the BNO055) but are not proper soft sensors.

Overall, this method is suitable to sense the shape and orientation of a soft body presenting a Piece-wise constant curvature behavior. Since the position is integrated from the base to the end effector, the end effector is the worst case scenario, but the shape recovery remains quite reliable. Also, since the sensors are absolute orientation sensors, we get by default the absolute orientation of the shape with a high reliability.

6.2 Possible applications

This method is suitable to sense the shape and orientation of a soft body presenting a Piece-wise constant curvature behavior.

The most obvious application is soft-continuum arms, such as the one that my

supervisor Daniel Bruder is developing (Fig. 6.1). Though, the advantages of the method (inexpensive, available off-the-shelf, easy to integrate) makes broader outreach possible, such as the use of IMUs arrays to create proprioceptive VR suits for example. Other fancy applications could be imagined, such as shape estimation of paragliding wings, and so on.

Future work could also investigate ways to make IMUs easier to integrate into soft robots. This could be done by mounting the IMUs on small custom PCBs, and by integrating an entire string of IMUs as a self-contained sensor.

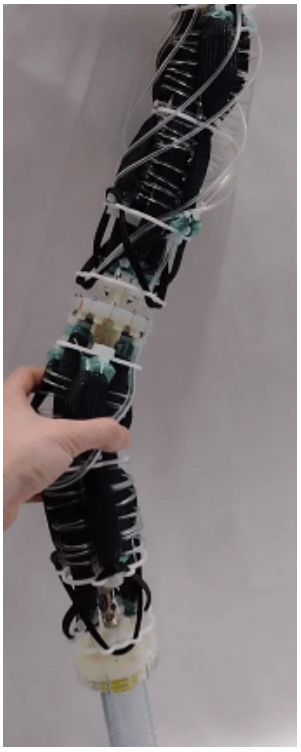


Figure 6.1: Daniel Bruder's soft arm, currently being developed. This arm will use this thesis content to implement the proprioception.

7 Conclusion

Soft robots are naturally compliant robots. They are well suited for human-robots interactions, wearables, safe and adaptive manipulation, and many more. This natural compliance makes proprioception challenging, and there is no silver bullet.

This project enlarged the offer for soft robotics proprioception, especially for large soft robots. Through this research, we intend to take soft robots off the labs, and to use them outdoor, without the help of motion capture. The inexpensive and integratable properties of the IMUs make them suitable for large scaled applications. This work is an opened door opening for deployable and safe soft robots systems.

8 Acknowledgements

First of all, I would like to warmly thank my direct supervisor and second author Dr. Daniel Bruder for the great researcher he is, his vision of the field, his intuition, and his hands-on approach to make hard problems easier. But also, I want to thank Dan for all of his support during these 6 months, his unalterable optimism and capacity to ask the right questions, always there when needed. It was truly a pleasure working with you.

I would also like to thank Prof. Robert Wood for the high-level vision feedback he provided to the project, and for welcoming me to his lab. I really appreciated how responsive and insightful Rob have been. The friendly and hard working lab atmosphere reflect perfectly his management style.

Thanks to Prof. Mohammed Bouri, for taking the time to supervise my thesis on the EPFL side. I truly appreciate how he took time to give me an opportunity to present at the lab meeting in March, in order to train and get feedback. I am extremely grateful for how he is dedicated to teaching, and for him accepting to supervise my thesis even though he's already very busy.

I would also like to thank Dr. Clark Teeple for all of the help he provided with prototyping, such as with the 3D printing machines, the silicon molding, questions about ROS, and so on. I learned a lot from you, not only from your answers but also from your approach.

Thanks to Dr. Patrick Hyun, Dr. Perrin Schiebel, and Jennifer Shum, for all of the thinking we had about a CPG-based controller for HAMR. I truly appreciated how you made yourself available, even in busy times, to discuss about this, review slides, this was a lesson of robotics passion.

Thanks to the entire Wood lab, for being so friendly, passionate, and helpful. This experience with you inspired me to push myself higher and I'll keep great memories of our ping-pong games! I would specifically like to thank Francisco, Pierre-Louis, Sandra and Bernadette, with whom I had such a great time during breaks and outside the lab.

Eventually, thanks to NASA's Space Technology Research Grants Program for funding my master's project.

9 Bibliography

References

[1] H. Wang, M. Totaro, and L. Beccai, "Toward perceptive soft robots: Progress and challenges," *Advanced Science*, vol. 5, no. 9, p. 1800541, 2018.

[2] Y.-L. Park, B.-R. Chen, and R. J. Wood, "Design and fabrication of soft artificial skin using embedded microchannels and liquid conductors," *IEEE Sensors journal*, vol. 12, no. 8, pp. 2711–2718, 2012.

[3] J. T. Muth, D. M. Vogt, R. L. Truby, Y. Mengüç, D. B. Kolesky, R. J. Wood, and J. A. Lewis, "Embedded 3d printing of strain sensors within highly stretchable elastomers," *Advanced materials*, vol. 26, no. 36, pp. 6307–6312, 2014.

[4] T. Helps and J. Rossiter, "Proprioceptive flexible fluidic actuators using conductive working fluids," *Soft robotics*, vol. 5, no. 2, pp. 175–189, 2018.

[5] M. C. Yuen, R. Kramer-Bottiglio, and J. Paik, "Strain sensor-embedded soft pneumatic actuators for extension and bending feedback," in *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pp. 202–207, 2018.

[6] K. C. Galloway, Y. Chen, E. Templeton, B. Rife, I. S. Godage, and E. J. Barth, "Fiber optic shape sensing for soft robotics," *Soft Robotics*, vol. 6, no. 5, pp. 671–684, 2019. PMID: 31241408.

[7] W. Zhuang, G. Sun, H. Li, X. Lou, M. Dong, and L. Zhu, "Fbg based shape sensing of a silicone octopus tentacle model for soft robotics," *Optik*, vol. 165, pp. 7–15, 2018.

[8] H. Zhao, J. Jalving, R. Huang, R. Knepper, A. Ruina, and R. Shepherd, "A helping hand: Soft orthosis with integrated optical strain sensors and emg control," *IEEE Robotics & Automation Magazine*, vol. 23, no. 3, pp. 55–64, 2016.

[9] I. Van Meerbeek, C. De Sa, and R. Shepherd, "Soft optoelectronic sensory foams with proprioception," *Science Robotics*, vol. 3, no. 24, p. eaau2489, 2018.

[10] M. Luo, E. H. Skorina, W. Tao, F. Chen, S. Ozel, Y. Sun, and C. D. Onal, "Toward modular soft robotics: Proprioceptive curvature sensing and sliding-mode control of soft bidirectional bending modules," *Soft Robotics*, vol. 4, no. 2, pp. 117–125, 2017. PMID: 29182091.

[11] S. Ozel, N. A. Keskin, D. Khea, and C. D. Onal, "A precise embedded curvature sensor module for soft-bodied robots," *Sensors and Actuators A: Physical*, vol. 236, pp. 349–356, 2015.

[12] W. Felt, K. Y. Chin, and C. D. Remy, "Contraction sensing with smart braid mckibben muscles," *IEEE/ASME Transactions on Mechatronics*, vol. 21, no. 3, pp. 1201–1209, 2015.

[13] N. Lazarus and S. S. Bedair, "Bubble inductors: Pneumatic tuning of a stretchable inductor," *AIP Advances*, vol. 8, no. 5, p. 056601, 2018.

[14] W. Felt, M. J. Telleria, T. F. Allen, G. Hein, J. B. Pompa, K. Albert, and C. D. Remy, "An inductance-based sensing system for bellows-driven continuum joints in soft robots," *Autonomous robots*, vol. 43, no. 2, pp. 435–448, 2019.

[15] J. C. Case, J. Booth, D. S. Shah, M. C. Yuen, and R. Kramer-Bottiglio, "State and stiffness estimation using robotic fabrics," in *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pp. 522–527, April 2018.

[16] M. C. Yuen, H. Tonoyan, E. L. White, M. Telleria, and R. K. Kramer, "Fabric sensory sleeves for soft robot state estimation," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 5511–5518, 7 2017.

[17] J. Tapia, E. Knoop, M. Mutný, M. A. Otaduy, and M. Bächer, "Makesense: Automated sensor design for proprioceptive soft robots," *Soft Robotics*, vol. 7, no. 3, pp. 332–345, 2020. PMID: 31891526.

[18] R. L. Truby, C. D. Santina, and D. Rus, "Distributed proprioception of 3d configuration in soft, sensorized robots via deep learning," *IEEE Robotics and Automation Letters*, vol. 5, pp. 3299–3306, April 2020.

[19] T. G. Thuruthel, B. Shih, C. Laschi, and M. T. Tolley, "Soft robot perception using embedded soft sensors and recurrent neural networks," *Science Robotics*, vol. 4, no. 26, p. eaav1488, 2019.

[20] N. Ahmad, R. A. R. Ghazilla, N. M. Khairi, and V. Kasi, "Reviews on various inertial measurement unit (imu) sensor applications," *International Journal of Signal Processing Systems*, vol. 1, no. 2, pp. 256–262, 2013.

[21] J. Hughes, F. Stella, C. D. Santina, and D. Rus, "Sensing soft robot shape using imus: An experimental investigation," in *International Symposium on Experimental Robotics*, pp. 543–552, Springer, 2020.

[22] O. D. Yirmibesoglu and Y. Menguc, "Hybrid soft sensor with embedded imus to measure motion," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 798–804, IEEE, 2016.

[23] D. Bruder and R. J. Wood, "The chain-link actuator: Exploiting the bending stiffness of mckibben artificial muscles to achieve larger contraction ratios," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 542–548, 2021.

[24] T. Foote, "tf: The transform library," in *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pp. 1–6, 2013.

[25] I. Robert J. Webster and B. A. Jones, "Design and kinematic modeling of constant curvature continuum robots: A review," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1661–1683, 2010.

[26] R. K. Katzschmann, C. D. Santina, Y. Toshimitsu, A. Bicchi, and D. Rus, "Dynamic motion control of multi-segment soft robots using piecewise constant curvature matched with an augmented rigid body model," in *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*, pp. 454–461, April 2019.

[27] Wikipedia contributors, "Quaternion — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Quaternion&oldid=1075949837, 2022. [Online; accessed 5-April-2022].

[28] M. Leeney, "Fast quaternion slerp," *International Journal of Computer Mathematics*, vol. 86, no. 1, pp. 79–84, 2009.

[29] Wikipedia contributors, "Inertial measurement unit — Wikipedia, the free encyclopedia," 2022. [Online; accessed 3-April-2022].

[30] Adafruit, "Adafruit bno055 absolute orientation sensor overview." https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview, 2022. [Online; accessed 7-April-2022].

[31] B. Sensortec, "Bno055 intelligent 9-axis absolute orientation sensor," *Bosch Sensortec, Baden-Württemberg, Germany*, pp. 1–106, 2016.

[32] Z. Mahdi, A. Ababou, N. Ababou, and R. Dumas, "Imu-based sensor-to-segment multiple calibration for upper limb joint angle measurement—a proof of concept," *Medical Biological Engineering Computing*, vol. 57, 08 2019.

[33] R. Mahony, T. Hamel, and J.-M. Pflimlin, "Nonlinear complementary filters on the special orthogonal group," *IEEE Transactions on Automatic Control*, vol. 53, no. 5, pp. 1203–1218, 2008.

[34] S. Madgwick *et al.*, "An efficient orientation filter for inertial and inertial/magnetic sensor arrays," *Report x-io and University of Bristol (UK)*, vol. 25, pp. 113–118, 2010.

[35] "Sensor Fusion Algorithms adafruit." https://learn.adafruit.com/how-to-fuse-motion-sensor-data-into-ahrs-orientation-euler-quaternions/sensor-fusion-algorithms. Accessed: 2022-4-4.

[36] " The Case of the Misguided Gyro ian beavers." https://www.analog.com/en/analog-dialogue/raqs/raq-issue-139.html. Accessed: 2022-4-4.

[37] F. Ferraris, U. Grimaldi, and M. Parvis, "Three-axis rate gyros and accelerometers," *Sensors and Materials*, vol. 7, no. 5, pp. 311–330, 1995.

[38] "Local Gravity Calculator." https://www.sensorsone.com/local-gravity-calculator/. Accessed: 2022-4-4.