Whole Body Kalman Filter

State estimation of a humanoid robot using nonlinear constrained optimisation

Arreguit O’Neill Jonathan Patrick

Professor
Supervisor

AuKe Ijspeert
Salman Faraji

Biorob
Laboratoire de biorobotique
The human robot as a system has a set of sensors and actuators. The controller reads sensor values and generate proper commands to perform a required task. In a model based control paradigm, the controller has either a kinematics or dynamics model of the robot to improve generated commands and make them matching more with the real dynamics of the robot. In this paradigm, having a precise model of the robot is valuable unless we can estimate the actual state of the robot based on incoming noisy sensor readings. The aim of this project is to use proper filters that reject the noise and estimate robotics state precisely. Although one can simply filter each individual sensor, this method is delay-full and make model-based control methods unstable. Thus we are interested in using more advanced methods to avoid delays. One possibility is Kalman filter, which takes model of the system into account. However one is encouraged to investigate other methods too. A candidate for this project is recommended to have knowledge about Dynamics model of a robot and C coding.
Abstract

In this project, the implementation of the state estimation of a coman (COMpliant HuMANoid) robot was studied. The information obtained from state estimation can typically be used for a controller or identification. Thus, accurate state estimation is one of the necessary steps before achieving a fully autonomous system. During the course of this project, the robot itself was not available at the laboratory. However, a simulation model was implemented in Webots and used in order to test different state estimation algorithms. The model of the coman robot included the different sensors found on the real robot such as an IMU at the level of the pelvis, encoders for each joint and force/torque sensors at each ankle. State estimation implies reading the sensor measurements and fusing them together in order to yield an optimal estimate of the robot’s relation to the world. Two separate scenarios were considered for this project. The first was that the feet of the robot were static, meaning they would not slip or tilt on the ground. As such, the positions and movements of each part of the robot could be computed from this fixed position using the measurement from the joints. The IMU and force/torque sensors were also used in order to yield a more accurate estimation. The goal of the second scenario was to consider slippage despite the lack of sensors capable of locating the robot’s global position. In this scenario, the usage of the IMU and force/torque sensors was considered critical in order to estimate the movement of the robot in the global frame. In particular, the idea was that the position drift caused by integration of the IMU could be limited by considering that the work produced by friction between the robot and the world can only be negative, and that the orientation of the estimated force at the position of the contact with the ground should be collinear to the estimated velocity of the robot at that point. The project was completed by first getting an overview of the techniques used by different groups working on the implementation of state estimation in robots, in particular for humanoids. It was decided that the most reasonable approach for both scenarios would be to use nonlinear constrained optimisation methods in order to fuse sensor data and obtain an optimal estimate. This approach allowed to make direct use of the physical and geometrical relations, while attempting to minimise the error. As such, different constrained optimisation algorithms were designed for the static scenario in order to build up iteratively to the dynamic scenario. The state estimation was then implemented for both scenarios. Furthermore, the complexity of the project required to design specific tools. In particular, a GUI environment for the visualisation and control of the robot data was implemented, as well as an automatic code generation script to generate code directly from the equations set. However, the implementation of the dynamic state estimation was not fully implemented during the time of this project. On the other hand, the state estimation designed for the static scenario was implemented successfully, which is the common approach considered by different labs working on state estimation with legged robots. This report discusses how constrained optimisation algorithms were setup and shows the results obtained. Further discussion about the interpretation of the results can also be found.

Acknowledgements

I would first like to thank Salman Faraji for the great support that he offered during the development of this project. The list of documentation provided in order to rapidly get up to date with the theory and current works as well as the numerous long talks were greatly appreciated. I would also like to thank Ludovic Righetti from the Max-Plank Institute, who did not hesitate to answer questions and offer helpful tips in the subject of state estimation. I would also like to gratefully thank Professor Auke Ijspeert and the members of the laboratory for the great work and enthusiasm that can be found at Biorob.
List of Figures

1.1 Project introduction .................................................. 1
2.1 Demonstration of slippage with the real robot ...................... 2
2.2 coman robot ............................................................ 5
2.3 Demonstration of static contact ...................................... 6
2.4 Demonstration of slippage ............................................ 7
2.5 Full robot model ......................................................... 9
2.6 Model of the coman robot’s foot on a slope ...................... 10
2.7 GUI - Control ............................................................ 23
2.8 GUI - Plot ............................................................... 24
2.9 Plot generation .......................................................... 24
3.1 Integration of IMU leading to drift .................................. 25
3.2 AO - SCIPY - SNOPT: Estimated velocities of the base ......... 25
3.3 AO - SCIPY - SNOPT: Estimated right foot CoP position ........ 26
3.4 AO - SCIPY - SNOPT: Estimated left foot CoP position ......... 26
3.5 AO - Inter: Comparison of estimated base position ............. 26
3.6 AO - Inter: Comparison of estimated base velocity ............. 27
3.7 AO - Inter: Comparison of estimated angular velocity ........ 27
3.8 AO - Inter: Comparison of estimated right foot CoP position 27
3.9 AO - Inter: Comparison of estimated left foot CoP position .... 28
3.10 Noise: Accelerometer ................................................. 29
3.11 Noise: Gyroscope ...................................................... 29
3.12 Noise: Magnetometer .................................................. 29
3.13 Noise: Right force sensor ............................................ 30
3.14 Noise: Left force sensor ............................................... 30
3.15 Noise: Right torque sensor .......................................... 30
3.16 Noise: Left torque sensor ............................................ 31
3.17 Noise: Estimation of the base velocity ............................ 31
3.18 Noise: Estimation of the right foot CoP position ................ 31
3.19 Noise: Estimation of the left foot CoP position ................ 32
3.20 AO - Full: Comparison of estimated base position ............. 32
3.21 AO - Full: Comparison of estimated base velocity ............. 33
3.22 AO - Full: Comparison of estimated base angular velocity ... 33
3.23 AO - Full: Comparison of estimated right foot CoP position ... 33
3.24 AO - Full: Comparison of estimated left foot CoP position .... 34

List of Tables

1 coman robot hardware specifications ................................ 5
2 State estimation implementation statistics .......................... 22
3 Noised sensors and estimated state standard deviations ........ 28

List of Abbreviations

• COMAN: COmpliant nuMANoid
• EKF: Extended Kalman Filter
• UKF: Unscented Kalman Filter
• IMU: Inertial Measurement Unit
• DOF: Degrees Of Freedom
• CoP: Centre of Pressure
• GUI: Graphical User Interface
• FFT: Fast Fourier Transform
• SLSQP: Sequential Least SQuares Programming
• STD: Standard deviation

List of Symbols

• $R_{\alpha}^\beta$: Rotation matrix for coordinate transformations
  - $R$: Rotation matrix
  - $\alpha$: Global frame
  - $\beta$: Local frame

• $\omega_{\alpha,\gamma}^\beta$: Rotational velocity
  - $\omega$: Rotational velocity
  - $\alpha$: Global frame
  - $\beta$: Frame of interest rotating in global frame.
  - $\gamma$: Frame in which the rotational velocity is expressed. Note that $\omega_{\alpha}^\beta$ means expressing the angular velocity in the local frame and $\omega_{\alpha}^\beta$ means expressing the angular velocity in the global frame. Moreover, $\omega_{\alpha,\alpha}^\beta = \omega_{\alpha}^\beta$ is used in this report.

• $S(\cdot)$: Skew matrix of a vector
  - In particular, it is extensively used for converting a cross-product into a matrix multiplication in this project.

• $y, \dot{y} & \ddot{y}$: Re-notation of the movement of the base, i.e. $x_{b}^{b}, \dot{x}_{b}^{b}, \ddot{x}_{b}^{b}$ respectively.

Further explanations of the mathematical operations for coordinate transformations can be found in Appendix A.1.
1 Introduction

In humanoid robots, estimating the state of the robot accurately is critical in order to perform tasks autonomously. The state of the robot includes all the information needed to know the robot’s relation to the world. It typically includes its position, posture, as well as how it is moving at a given time. As the control requires the state, it is necessary to have an accurate estimation in order for the robot to perform advanced tasks.

In this project, the implementation of the state estimation of a Coman (COmpliant HuMANoid) robot is studied. The Coman robot has a set of sensors and actuators. It is quite a complex humanoid capable of delivering fast reactions when necessary, while still offering the capacity to perform small precise movements for humanoid tasks. State estimation implies computing an estimate of the state of the robot using the available measurements from sensors and the different hypothesis determined. This means using on-board sensors and exploiting physical relations in order to obtain the best estimate of the robot at a given time.

The implementation was largely inspired from Salman Faraji’s work [1], where the state estimation was obtained using optimisation techniques. In his work, the main hypothesis made was that the contact between the ground and the robot was static, i.e. the feet of the robot on the ground would not slip or tilt. However, slippage was observed to occur with the Coman robot when performing rapid movements. This may also for example happen on a slope, where the robot could start sliding down or tilt. As such, the next logical step was to explore the implementation of state estimation when considering dynamical contacts. Moreover, an objective was to achieve this without bringing any hardware modifications to the robot, i.e. no additional sensors were to be added.

To implement the state estimation, the Kalman filter was first considered. However, the plain Kalman filter was found to not be ideal for this project. Instead, nonlinear constrained optimisation algorithms were used, which work similarly to the Kalman filter, while also taking into account constraints. The proposed framework allowed to fuse the sensor measurements with the physical and geometrical constraints. This can be done recursively, which is suitable for real-time computing on the robot, similarly to the Kalman filter. The physical constraints included using Newton’s laws of dynamics, friction hypothesis between the robot’s feet and the ground, as well as work. By fusing all the sensor measurements together through these constrains, it was expected that it would be possible to compute the full state of the robot.

As the state estimation was already known to work for static contact, the goal was to compare results between the static and dynamic algorithms to verify if considering dynamic contacts was possible for state estimation on legged robots not including global positioning sensors.

In this report, the state-of-the-art techniques used to provide state estimation are presented. Furthermore, the methods used in order to design and implement the state estimation are discussed in Section 2. The results obtained can then be found in Section 3. Finally, the follow-up to these results is discussed in Section 4.

![Coman robot](image1.png) ![Model of the Coman robot as seen in Webots](image2.png)

*Figure 1.1: The Coman robot. A model of this robot was used in simulation in order to develop and test the state estimation algorithms. The simulation was done using Webots 7.2.1*
2 Models and methodology

2.1 Demo of slippage

The coman robot was observed to slip when performing certain tasks. An example of slippage is illustrated in Figure 2.1. In this case, the slippage would occur after performing multiple angular rotation movements of the torso. Position 1 shows the robot at its original Position. Position 2 then shows by how much the robot would rotate over the course of about half a second. A slight displacement of the foot may already be observed in this Position. A total of 12 right to left Positions were performed over the course of 12 seconds by performing a movement going from Position 1 to Position 2 and back. One may observe that the foot of the robot at the end of these movements would make the robot move to Position 3, which is displaced compared to Position 1. This was caused by slippage between the foot of the robot and the ground. The goal of this project is to find a method to consider this slippage in the state estimation using the available sensors.

![Figure 2.1: Demonstration of slippage with the real robot when performing rapid movements.](image)

2.2 State of the art

The work done in this project is mostly based on the implementation by Salman Faraji found in [1], where the practical implementation of inverse dynamics on a humanoid robot was studied, namely torque tracking, state estimation and control. This work was also produced using a coman robot, where the state estimation was completed using optimisation techniques with the available IMU, joint measurements and force/torque sensors, similarly to a Kalman filter. This implementation was studied in this project and reimplemented, which is discussed in Sections 2.5 and 2.8.1. The main hypothesis for this state estimation was that the contact between the feet or the hands with the world was static.

This approach has also been used by other labs, such as [2], who directly used an Extended Kalman filter in order to estimate the state of the robot. In their work, they proposed to use full-body dynamics for the state estimation of an Atlas robot. They mentioned the possibility of decoupling the full body state vector into several independent state vectors. By decoupling, they explained that the state vectors could be estimated using a steady state Kalman filter, and demonstrated this approach could lead to the robot walking on both flat ground and rough terrain. Furthermore, they explained the functionality of the Kalman filter quite nicely:

"Kalman Filter is a recursive filter that estimates the internal state of a linear dynamic system from a series of noisy measurements. To handle nonlinearity, the Extended Kalman Filter (EKF) and later the Unscented Kalman Filter (UKF) were invented. The EKF linearizes the nonlinear dynamics at the
current mean estimate, and propagates the belief or information state covariance the same way as the KF. The UKF samples around the mean estimate based on the state covariance to create sampling points (sigma points), and propagates the mean and covariance of the belief or information state using the sigma points. We list the discrete time EKF equations below for future reference. The EKF equations are given in two steps:

**Prediction step:**

\[
x_k^- = f(x_{k-1}^-, u_{k-1}) \tag{2.1}
\]

\[
P_k^- = F_k P_{k-1}^+ F_k^T + Q_k \tag{2.2}
\]

**Update step:**

\[
y_k = h(x_k^-) \tag{2.3}
\]

\[
\Delta y_k = z_k - y_k \tag{2.4}
\]

\[
S_k = H_k P^- k H_k^T + R_k \tag{2.5}
\]

\[
K_k = P^- k H_k^T S_k^{-1} \tag{2.6}
\]

\[
\Delta x_k = K_k \Delta y_k \tag{2.7}
\]

\[
x_k^+ = x_k^- + \Delta x_k \tag{2.8}
\]

\[
P_k^+ = (I - K_k H_k) P_k^- \tag{2.9}
\]

The subscript \( k \) is the step index, the superscript “-” and “+” represent before and after the measurement update, capital letters are matrices, and lower case letters stand for vectors. \( F_k \) and \( H_k \) are Jacobian matrices of \( f \) and \( h \) linearized around the mean estimate. \( F_k \) is the state transition matrix, \( H_k \) is the observation matrix, \( P \) is the state covariance, and \( K_k \) is the Kalman Gain. \( z_k \) is the actual measurement, \( y_k \) is the predicted measurement, and \( \Delta y_k \) is called the innovation or measurement residual. In this recursive formulation, one expensive operation is to compute \( S_k^{-1} \) in Equation (2.6). If \( F_k \) and \( H_k \) are computed by numeric differentiation at each recursion, they are also computationally expensive. In the linear KF settings, if \( F_k \), \( H_k \), \( Q_k \) and \( R_k \) are time invariant (constant), then \( P_k \) and \( K_k \) will converge to their steady state values. It is uncommon for an EKF to have time invariant \( F_k \) and \( H_k \). If we assume they are constant over a certain period of time or some states, we could formulate the recursive EKF problem as a steady state EKF problem.\(^2\)

These algorithms are not only used for humanoid robots, but can also be used with other types. For example \(^3\) used an EKF for the state estimation of a quadruped robot. However, most laboratories seem to base the state estimation on the idea that there is little to no slippage. Indeed, most seem to attempt to actually avoid slippage, as slippage could introduce disturbances that would be difficult to model for a Kalman filter. This is either done by designing hardware accordingly such as specific shoes, or by avoiding aggressive movements.

However, \(^4\) proposed that a Kalman filter may not always be suitable for humanoid robots. Indeed, the Kalman filter requires prediction, but the presence of the un-modelled noise may lead to the observed dynamics to differ from the expected dynamics. Instead, the proposition was to use simplified models for the sensor data, which was shown to obtain accurate enough estimation for humanoid balancing, even with the presence of disturbances. In particular, they make use of the sensors to find the Centre of Pressure (CoP) for the contact of the feet with the ground. This is typically done by including force and torque sensors on the foot of the robot and allows the controller to react appropriately, such as when walking on rough terrain. Such sensors are also available on the coman robot.

Furthermore, some implementations directly rely on optimisation techniques. This was for example the case for \(^5\), who used quadratic programming on an Atlas humanoid robot. The conversion from a simple Kalman filter into an optimisation problem is addressed by \(^6\), which even showed how a nonlinear constrained optimisation problem can replace a Kalman filter.
Optimisation methods are typically written as follows:

$$\min_{x} f(x)$$

s.t.

$$g(x) = 0$$

$$h(x) \leq 0$$

(2.10)

where $f(x)$ is the objective function to minimise, $g(x)$ are the hard constraints and $h(x)$ are the soft constraints. Optimisation techniques usually rely on minimising the error of each sensor such that the given constraints are respected. A cost is usually given to each error, where sensors that are known to yield more accurate measurements may have a higher cost and would be further minimised. This is similar to the Kalman filter, in which the covariance matrices of the sensors are taken into account. As proposed by [1], using optimisation becomes equivalent to Kalman filtering if the costs for the errors are considered to be the inverse of the error’s covariance matrix for each equation, typically written as the constraints of the optimisation problem. Moreover, by minimising the square of the errors, the constrained optimisation will also be expected to yield optimal results for Gaussian noise. This approach was considered the most suitable in order implement the state estimation of the coman robot.

2.3 Proposition

In this project, the goal is to find the state of the robot. Considering rigid bodies, this would include the positions, velocities and accelerations of each point of the robot. These 6 DOF positions/orientations and their first and second derivatives of each point of the body are called "Cartesian states". These Cartesian states would be intermediate variables. By estimating the position of a specific point of the robot, named "base", the Cartesian states of any point on the body could be related to this point using internal kinematics, which depend on the measured joint angles. The base was chosen to be located at the position of the IMU sensor on the pelvis and would be oriented in the same way as this IMU.

Moreover, another objective was to determine the external forces and their positions on the feet. This was to be done using the sensors measurements. As the robot is bound to follow the laws of physics, the forces acting on the robot can be linked to their positions using physical and geometrical equations. If the sensors were perfect, then the state of the robot could be obtained by computing the equations with the measurements. However, the challenge is that sensors are noisy, have an offset and can drift over time. As such, the state must be computed by fusing all measurements and equations together in order to obtain the most likely state. From the current work done in state estimation, the hot topic is the use of optimisation techniques, as mentioned previously in Section 2.2. As such, the state would be computed by assigning a weight to each sensor and minimise the error, where the reliable sensors would be favoured in the case of inconsistent measurements. This ends up providing a similar algorithm to the Kalman filter. However, the Kalman filter computes the optimal solution for a linear problem. Furthermore, the Kalman filter would also require the system model for the prediction step. In the case of this project, only the sensor model would be used, as a prediction step would require a full body system identification. However, the proposed algorithm makes it quite simple to add this prediction if required. This would be expected to improve results, as long as the system model would be sufficiently accurate and model disturbances. This is further discussed in Section 2.6.

As such, the IMU was to be used in order to give an estimation of the movement of the base. As this integration was expected to be prone to drift, this information needs to be fused with the measurements of the ankle force sensors. The physical and geometrical relations are discussed in Section 2.5. The optimal state could then be computed using nonlinear constrained optimisation techniques, discussed in Section 2.7.
2.4 Materials

2.4.1 coman

This project was undertaken using many tools at disposal. Unfortunately, the coman robot itself was not available during the course of the project. Instead, a model of the coman robot was simulated in Webots 7.2.1. The following description was taken from the official website [7].

"The COMAN humanoid robot [...] is being developed within the AMARSI European project which aims to achieve a qualitative jump toward rich motor behaviour in robotic systems, rigorously following a systematic approach in which novel mechanical systems with passive compliance, control and learning solutions will be integrated."

 [...] 

The COMAN robot is 95cm tall, weighs 31kg and has 25 DOF. Its mechanical components are made from titanium alloy, stainless steel and aluminium alloy, giving it good physical robustness. Its modular joint design uses brushless, frameless DC motors, Harmonic Drive gears and series elastic elements. Leg, waist and shoulder joints have a peak torque capability of 55Nm. Custom torque sensors are integrated into every joint to enable active stiffness control and 6-DOF sensors are included at the ankle to measure ground reaction forces. COMAN can walk and balance using inertial sensors in the pelvis and chest, and its series elastic joint design makes it robust against impacts and external disturbances. COMAN is fully power autonomous. The torso contains a dual core Pentium PC104, onboard battery and battery management system giving up to 2.5 hours continuous operation."

<table>
<thead>
<tr>
<th>coman robot hardware characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
</tr>
<tr>
<td>Weight</td>
</tr>
<tr>
<td>DOF</td>
</tr>
<tr>
<td>Mechanical components materials</td>
</tr>
<tr>
<td>Joints actuation components</td>
</tr>
<tr>
<td>Leg, waist &amp; shoulder joints</td>
</tr>
<tr>
<td>peak torque capability</td>
</tr>
<tr>
<td>Active stiffness control</td>
</tr>
<tr>
<td>Ankle sensors</td>
</tr>
<tr>
<td>Inertial sensors positions</td>
</tr>
<tr>
<td>On-board computer</td>
</tr>
<tr>
<td>Battery expected autonomy</td>
</tr>
</tbody>
</table>

Table 1: coman robot hardware specifications obtained from the description found in [7]. The IMU used in the coman robot is the 3DM-GXØ-25 [8]. Information about the force sensors can be found in [9]. Moreover, further details about the actuators can also be found in [10].

For the coman robot model used in simulation, the sensors were the IMU at the level of the the pelvis and the force sensors found in each foot. The joint angles were also considered. However, no noise was implemented in the simulation. This makes it simpler to design and test algorithms, but also makes it rather difficult to know if it would work on the real robot with noisy sensors. Implementing a model of the noise was considered, but the idea was abandoned as an implementation was not expected to fully reflect reality.
2.4.2 Software

The project was fully developed on a computer with Linux Debian 8. It is expected to be portable to other Linux platforms as long as the following software presented in this section can be obtained. It may also be possible to be ported on Mac and Windows, but is expected to be more tedious due to the coman framework relying on different typical Linux packages. Further explanations about installation can be found in Appendix B.

The coman robot was fully simulated using Webots 7.2.1. The framework was developed at Biorob and used for [1]. In particular, it is possible to connect the real coman robot to Webots, meaning that algorithms that work in the simulation could easily be exported. Obviously, the simulation may not be perfect and the results obtained with the proposed algorithms may not reflect exactly what would be achieved with the real robot, but are still expected to be quite similar. In order to test the algorithms in different conditions, multiple simulation scenarios were provided. In particular two main scenarios were used during the development of this project. The first, illustrated in Figure 2.3, was a scenario where the robot would rotate its body while keeping a static contact with the ground. This scenario would make the coman robot perform similar movements to those discussed in Section 2.1. This scenario allowed to compare results between the previous implementation of state estimation and the one developed during this project and verify that the new software libraries used would work properly. The second simulation scenario provided was one where the feet of the coman robot would slip on the ground. This scenario is illustrated in Figure 2.4 and was not perfectly stable, as the controller was not yet implemented for handling slippage. Moreover, the simulation of friction is not expected to reflect measurements that are experienced in reality. Still, this scenario would allow to simulate the slippage of the feet of the robot for a few seconds before it would fall and was still expected to be good enough to test the proposed algorithms.

The default movement used for this project with the static contact was one which would make the coman robot both move up and down and rotate along the gravity axis as illustrated in the series of images. Position 1 shows the robot facing slightly to its right and at its highest head position. Position 2 shows the robot in movement going from position 1 to position 3. Notice how the robot starts bending its legs as well. Position 3 shows the robot at it’s lowest position and facing straight with respect to it’s feet. The robot would then return to the original position 1 and repeat the process. An iteration of this process from one position and back would last 1 second in simulation time.

Figure 2.3: Demonstration of static contact with the ground during movement in Webots. The foot does not slip or tilt, meaning this it keeps a fix position in the world. This position can then be used as a reference in order to compute the positions and velocities of each robot part using the joint measurements. However, the encoders are known to have a limited resolution, meaning the accuracy of the measurement is decreased after each encoder. This low resolution means it necessary to fuse the data to other sensors in order to yield a more accurate estimation. The default movement used for this project with the static contact was one which would make the coman robot both move up and down and rotate along the gravity axis as illustrated in the series of images. Position 1 shows the robot facing slightly to its right and at its highest head position. Position 2 shows the robot in movement going from position 1 to position 3. Notice how the robot starts bending its legs as well. Position 3 shows the robot at its lowest position and facing straight with respect to it’s feet. The robot would then return to the original position 1 and repeat the process. An iteration of this process from one position and back would last 1 second in simulation time.
Figure 2.4: Demonstration of slippage with the ground during movement in Webots. The robot starts in position 1, where the pressure on the feet would be spread equally. This is illustrated by the green lines originating from each foot corner. Position 2 shows the robot shifting its weight to its right leg, with the left leg still in contact with the ground surface. Position 3 then shows the left foot of the robot to be at a different place compared to Position 2. The simulation would be capable of making the foot slide for about 10 seconds before falling down due to simulation instability. The left foot would slide from left to right 7 times before this would occur.

This simulation from the previous work used for [1] was mainly implemented in C++. As such, a part of the project was developed in C++ using multiple libraries, mainly:

- **Eigen** [11, 12] - "Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.'

- **SNOPT** [13, 14] - "SNOPT is a general-purpose system for constrained optimization. It minimizes a linear or nonlinear function subject to bounds on the variables and sparse linear or nonlinear constraints. It is suitable for large-scale linear and quadratic programming and for linearly constrained optimization, as well as for general nonlinear programs.'

Eigen was already included in the provided simulation environment. It was used in order to easily manipulate vectors and matrices in an efficient manner. SNOPT was not used previously and would come to replace the previous library used for optimisation which was CVXGEN. SNOPT has the advantage of being capable of solving nonlinear constrained optimisations problems.

Python 3.4 was also used during the development of the project, which included using the following typical libraries:

- **Numpy** [15, 16] - 'NumPy is the fundamental package for scientific computing with Python. It contains among other things: a powerful N-dimensional array object, sophisticated (broadcasting) functions, tools for integrating C/C++ and Fortran code, useful linear algebra, Fourier transform, and random number capabilities.'

- **Scipy** [17, 18] - 'SciPy (pronounced "Sigh Pie") is open-source software for mathematics, science, and engineering. It includes modules for statistics, optimization, integration, linear algebra, Fourier transforms, signal and image processing, ODE solvers, and more.'

- **SymPy** [19, 20] - 'SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python and does not require any external libraries.'

- **PySide** [21, 22] - 'Qt is a cross-platform application framework from Qt Software (owned by Nokia). It features a large number of libraries providing services like network abstraction and XML handling, along with a very rich GUI package, allowing C++ developers to write their applications once and
run them unmodified in different systems. PySide aims to provide Python developers access to the Qt libraries in the most natural way."

- PyQtGraph [23, 24] - "PyQtGraph is a pure-python graphics and GUI library built on PyQt4 / PySide and numpy. It is intended for use in mathematics / scientific / engineering applications. Despite being written entirely in python, the library is very fast due to its heavy leverage of numpy for number crunching and Qt’s GraphicsView framework for fast display. PyQtGraph is distributed under the MIT open-source license."

- Matplotlib [25, 26] - "Matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB or Mathematica), web application servers, and six graphical user interface toolkits."

Python was used for manipulating symbolic mathematics using Sympy. This was useful for directly studying the equations and automatically generating the corresponding code for the optimisation algorithms, discussed in Section 2.8.1.

Python was also directly embedded into the C++ code such that it could be launched and called from the controller algorithm. The reason why to embed Python was mainly to monitor the controller data. This included visualising the data online beside the simulation using PyQtGraph, while also offering the possibility of manipulating certain controller parameters through a GUI implemented with PySide. Plots using Matplotlib of the results could also be directly generated and manipulated from this GUI. Moreover, certain algorithms were also tested directly from this embedded Python to speed up prototyping and testing, which was done using the Scipy package. This GUI is presented in more detail in Section 2.8.2.

### 2.5 Sensor Model

In order to design a state estimation algorithm, a model of the robot must first be built. To do so, the geometrical and physical relations must be studied. This part is heavily based on [1]. Further information about coordinates transformations mathematics can be found in Appendix A.1.

#### 2.5.1 Body

In the coman robot, all joint angles are measured. This means that the position of each joint could be estimated using these joint angles as long as the position and orientation of a point on the robot is known. The Cartesian states also include velocities and accelerations. These states could be computed in a similar fashion by numerically deriving the joint angles. The base could then be linked to the foot using these measurements. Furthermore the Cartesian states can be given according to different frames defined as follows. Similarly to [1], the general frame of the world was referred to as the world frame $i$. However, the Cartesian states are obtained by linking them to the base. As such, a general frame that the Cartesian points would be related to through the joint angles would be denoted the base frame $b$. When comparing to Figure 2.5, this frame would be found around the position $C$. In particular, all IMU measurements are given in this base frame. Each foot also has multiple frames that are discussed in Section 2.5.3, which would include the centre of mass frame, the foot’s contact surface polygon frame and the force/torque sensor frame. These frame would be found around positions $P_R$ and $P_L$ of Figure 2.5, also details the use of static contact to compute the Cartesian states.
2.5.2 Base

This IMU allows to give a first estimation of the dynamics of the base by integrating or differentiating from measurements. The following equations show the information that can be extracted:

**Accelerometer:**

\[
\begin{align*}
    x_b^1(t) &= x_b^1(t - \Delta t) + \dot{x}_b^1(t)\Delta t \\
    \dot{x}_b^1(t) &= \dot{x}_b^1(t - \Delta t) + \ddot{x}_b^1(t)\Delta t \\
    \ddot{x}_b^1(t) &= R_{ib}^1 g_{imu}^b(t) - g^1
\end{align*}
\]

**Gyroscope:**

\[
\begin{align*}
    \phi_b^1(t) &= \phi_b^1(t - \Delta t) + \omega_b^1(t)\Delta t \\
    \omega_b^1(t) &= R_{ib}^1 \omega_{imu}^b(t) \\
    \alpha_{b}^1(t) &= \omega_b^1(t) - \omega_b^1(t - \Delta t)
\end{align*}
\]

**Magnetometer:**

\[
\begin{align*}
    \phi_{mag}^1(t) &= R_{ib}^1 \phi_{imu}^b(t) \\
    \omega_b^1(t) &= \phi_b^1(t) - \phi_b^1(t - \Delta t) \\
    \alpha_{b}^1(t) &= \omega_b^1(t) - \omega_b^1(t - \Delta t)
\end{align*}
\]

The position of the robot in the world frame can be obtained by integrating the accelerometer data while knowing its orientation in the world frame. However, it is important to keep in mind that the integration causes drift due to measurement noise and offsets, originating from imperfect physical sensors. As such, the IMU sensor cannot provide an accurate measurement of the position and movement over long periods of time, as the measurement error will also be integrated.
2.5.3 Foot

The coman robot’s foot includes a force and torque sensor placed at the position of the ankle. This allows to measure forces and moments between the foot and the rest of the body. By considering that the only forces applied on the foot are caused by gravity, the ground reaction, and the forces measured by the sensor, the foot can be isolated into an independent system, as illustrated in Figure 2.6. The following model and mathematics were all written for the right foot, although the equivalent equation for the left foot can be obtained similarly.

![Figure 2.6: Model of the coman robot’s right foot on a slope as proposed by [1].](image)

With this model, the right foot is defined as the centre of mass frame, $c_r$ is defined as the centre of the polygon contact surface of the foot with the ground, and $s_r$ is defined as the force/torque sensor frame. As such, $x_{sh_r}$ is defined as the position of the centre of mass of the foot. Furthermore, it is proposed that $R_{sh_r}^i = R_{sr}^i = R_{cr}^i$ to ease coordinate transformations between these frames. The state of the robot must include its relation to the ground. As such, the Contact of Pressure $CoP$ models the single point where the integral of the pressure applies onto the foot by the ground forces. This $CoP$ has three characteristics: its position $x_{CoP}^i$, forces $f_{CoP}^i$ and torques $m_{CoP}^i$. All of which are three dimensional vectors. However, note that the $CoP$ is expected to always be contained in the bottom surface of the foot, which renders the position $x_{CoP}^i$ to be a problem with only 2 unknowns instead of 3.

At first, the foot was considered to be static. If this hypothesis is followed, the sum of forces and moments applied on the foot respect the following equations according to Newton’s laws:

$$\sum f_{foot}^i = f_{sr}^i + f_{gravity}^i + f_{CoP}^i = 0 \quad (2.14)$$

$$\sum m_{foot}^i = m_{sr}^i + (x_{sr}^i \times f_{sr}^i) + (x_{sh_r}^i \times f_{gravity}^i) + m_{CoP}^i + (x_{CoP}^i \times f_{CoP}^i) = 0 \quad (2.15)$$

These equations are written in the global world frame. However, the sensors on the coman robot measure values in the local frame. As such, Equations (2.14) and (2.15) can be rewritten using local frames:

$$\sum f_{foot}^i = f_{sr}^i + f_{gravity}^i + f_{CoP}^i$$

$$= R_{sh_r} f_{sh_r} + R_{sr} f_{gravity} + R_{sh_r} f_{CoP}$$

$$= R_{sh_r} (r_{sr} + r_{sh_r} g + r_{sh_r} C_{CoP}) = 0 \quad (2.16)$$

$$\Rightarrow r_{sr} + (R_{b} R_{sh_r})^{-1} r_{gravity} + r_{sh_r} C_{CoP} = 0$$
\[ \sum \mathbf{m}_{\text{foot}}^i = \mathbf{m}_{s_r}^i + \mathbf{S}(\mathbf{x}_{s_r}^i)\mathbf{f}_{s_r}^i + \mathbf{S}(\mathbf{x}_{s_r}^i)\mathbf{f}_{\text{gravity}}^i + \mathbf{m}_{\text{CoP}}^i + \mathbf{S}(\mathbf{x}_{\text{CoP}}^i)\mathbf{f}_{\text{CoP}}^i \]

\[ = \mathbf{R}_i^b \mathbf{R}_s^{b_r} \mathbf{m}_{s_r}^{b_r} + \mathbf{S}(\mathbf{R}_i^b \mathbf{R}_s^{b_r} \mathbf{x}_{s_r}^{b_r}) \mathbf{R}_i^b \mathbf{R}_s^{b_r} \mathbf{f}_{s_r}^i + \mathbf{S}(\mathbf{R}_b^i \mathbf{R}_s^{b_r} \mathbf{x}_{s_r}^{b_r}) \mathbf{f}_{\text{gravity}}^i \]

\[ + \mathbf{R}_i^b \mathbf{R}_s^{b_r} \mathbf{m}_{\text{CoP}}^{b_r} + \mathbf{S}(\mathbf{R}_i^b \mathbf{R}_s^{b_r} \mathbf{x}_{\text{CoP}}^i) \mathbf{R}_i^b \mathbf{R}_s^{b_r} \mathbf{f}_{\text{CoP}}^i \]

\[ = \mathbf{R}_i^b \mathbf{R}_s^{b_r} \mathbf{m}_{s_r}^{b_r} + \mathbf{R}_i^b \mathbf{R}_s^{b_r} \mathbf{S}(\mathbf{x}_{s_r}^i) \mathbf{f}_{s_r}^i \]

\[ + \mathbf{R}_i^b \mathbf{R}_s^{b_r} \mathbf{m}_{\text{CoP}}^{b_r} + \mathbf{R}_i^b \mathbf{R}_s^{b_r} \mathbf{S}(\mathbf{x}_{\text{CoP}}^i) \mathbf{f}_{\text{CoP}}^i = 0 \]

\[ \Rightarrow \mathbf{m}_{s_r}^{b_r} + \mathbf{S}(\mathbf{x}_{s_r}^{b_r}) \mathbf{f}_{s_r}^i + \mathbf{m}_{\text{CoP}}^{b_r} + \mathbf{S}(\mathbf{x}_{\text{CoP}}^i) \mathbf{f}_{\text{CoP}}^i = 0 \] \hspace{1cm} (2.17)

For a static system, the force \( \mathbf{f}_{\text{CoP}}^i \) can be computed in order to satisfy Equation (2.16), as \( \mathbf{f}_{s_r}^i \) and \( \mathbf{f}_{\text{gravity}}^i \) can be measured. However, note that the orientation of the foot must be known in order for the gravity to be applied correctly, i.e. orienting it in the foot frame. For Equation (2.17), both the position \( \mathbf{x}_{\text{CoP}}^i \) and the torque applied by the ground onto the foot \( \mathbf{m}_{\text{CoP}}^i \) are unknown. However, the structures of the CoP variables can be hypothesised as follows:

\[ \mathbf{f}_{\text{CoP}}^{b_r} = \begin{pmatrix} f_1^i \\ f_2^i \\ f_3^i \end{pmatrix}, \quad \mathbf{m}_{\text{CoP}}^{b_r} = \begin{pmatrix} 0 \\ 0 \\ \tau \end{pmatrix} \quad \text{and} \quad \mathbf{x}_{\text{CoP}}^i = \begin{pmatrix} x_1 \\ x_2 \\ 0 \end{pmatrix} \] \hspace{1cm} (2.18)

Indeed, after obtaining \( \mathbf{f}_{\text{CoP}}^{b_r} \) from equation (2.16), both \( \mathbf{x}_{s_r}^i \) and \( \mathbf{m}_{\text{CoP}}^{b_r} \) can be computed from equation (2.17) as:

\[ \mathbf{x}_{\text{CoP}}^i \times \mathbf{f}_{\text{CoP}}^{b_r} = \begin{pmatrix} x_2 f_3 \\ -x_1 f_3 \\ x_1 f_2 - x_2 f_1 \end{pmatrix} \] \hspace{1cm} (2.19)

This means that \( \mathbf{x}_{\text{CoP}}^i \) can first be computed using the two first rows of Equation (2.17). \( \mathbf{m}_{\text{CoP}}^{b_r} \) can then be computed using the last row. These hypothesis are found to be true as long as all the contact points on the foot are applied on the surface polygon on the bottom of the foot. As such, these equations are expected to be solvable when given to an optimisation algorithm.

Next, since the feet are static, we also know that the positions and orientations of each foot are fixed, meaning that the velocities and angular velocities are equal to zero. The following equations relate the world frames to the local frames when considering these relations:

\[ \mathbf{x}_{s_r, \text{fix}}^i = \mathbf{x}_b^i + \mathbf{R}_b^{i} \mathbf{x}_{s_r}^{b_r} \]

\[ \dot{\mathbf{x}}_{s_r}^i = \frac{d}{dt} (\mathbf{x}_b^i + \mathbf{R}_b^{i} \mathbf{x}_{s_r}^{b_r}) \]

\[ = \frac{d\mathbf{x}_b^i}{dt} + \frac{d\mathbf{R}_b^{i-b}}{dt} \mathbf{x}_{s_r}^{b_r} + \mathbf{R}_b^{i-b} \frac{d\mathbf{x}_{s_r}^{b_r}}{dt} \]

\[ = \dot{\mathbf{x}}_b^i + \mathbf{R}_b^{i-b} \mathbf{S}(\omega_{s_r}^{i-b}) \mathbf{x}_{s_r}^{b_r} + \mathbf{R}_b^{i-b} \mathbf{x}_{s_r}^{b_r} = 0 \]

\[ \Rightarrow \mathbf{R}_{s_r, \text{fix}}^i = \mathbf{R}_b^{i-b} \mathbf{R}_{s_r}^{b_r} \]

\[ \omega_{s_r}^{i-b} = \mathbf{R}_b^{i-b} (\omega_{b}^{i-b} + \omega_{s_r}^{b_r}) = 0 \] \hspace{1cm} (2.23)

Note that \( \omega_{b}^{i-b} \) is given in the local base frame as the gyroscope inside the IMU directly measures this value. Moreover, \( \mathbf{x}_{\text{CoP}}^i \) must satisfy \( \mathbf{x}_{\text{CoP}}^i \in \text{foot’s contact surface} \) to assure static contact. As such, if equations (2.14) and (2.15) cannot satisfy this constraints, then the robot’s foot is not expected to be in a static regime. Finally, note that these equations could also be used if the feet of the robot were moving at constant linear and angular velocities, i.e. sliding at a constant speed for example. However, this kind of movement is unexpected, as movement of the feet will most likely also include accelerations.
Next, when considering dynamic contact, the foot must satisfy the following equations. The first equations (2.24) and (2.25) correspond to the sum of forces and sum of moments respectively. They are of course related to the linear and angular accelerations.

\[
\sum f_{\text{foot}}^i = f_{s_r}^i + f_{\text{gravity}}^i + f_{\text{CoP}}^i = M_{sh_r} x_{sh_r}^i \tag{2.24}
\]

\[
\sum m_{\text{foot,sh}}^i = m_{s_r}^i + (x_{s_r}^i - x_{sh_r}^i) \times f_{s_r}^i \\
+ (x_{sh_r}^i - x_{sh_r}^i) \times f_{\text{gravity}}^i \\
+ m_{\text{CoP}}^i + (x_{\text{CoP}}^i - x_{sh_r}^i) \times f_{\text{CoP}}^i \\
= I_{sh_r} \omega_{sh_r}^i \tag{2.25}
\]

Next, the work caused by friction with the ground can only dissipate energy and must be negative. Thus, the following equations can be written:

\[
f_{\text{CoP}}^i T x_{\text{CoP,fix}}^i \leq 0 \tag{2.26}
\]

\[
m_{\text{CoP}}^i T \omega_{\text{CoP,fix}}^i \leq 0 \tag{2.27}
\]

Furthermore, the ground reaction can also be decomposed into the sum of two vectors, with a first vector \( f_{\text{CoP,\perp}}^i \) included inside the foot's contact surface, and the second vector \( f_{\text{CoP,\parallel}}^i \) perpendicular to the contact surface. This decomposition would result in the following collinear and perpendicular relations with respect to velocities:

\[
f_{\text{CoP,\perp}}^i + f_{\text{CoP,\parallel}}^i = f_{\text{CoP}}^i \tag{2.28}
\]

\[
f_{\text{CoP,\perp}}^i T f_{\text{CoP,\parallel}}^i = 0 \tag{2.29}
\]

\[
f_{\text{CoP,\parallel}}^i \times \dot{x}_{\text{CoP,fix}}^i = 0 \tag{2.30}
\]

\[
m_{\text{CoP}}^i \times \omega_{\text{CoP,fix}}^i = 0 \tag{2.31}
\]

Note that the \( \dot{x}_{\text{CoP,fix}}^i \) and \( \omega_{\text{CoP,fix}}^i \) from equations (2.30) and (2.30) respectively correspond to the velocities of the point of the foot at which is positioned the CoP at a certain time, and the movement of the CoP itself. Similary to [1], the equations written above are all in the world frame and can be rewritten in the local frames as follows.
Sum of forces:

$$\sum f^i_{\text{foot}} = f^i_{s_r} + f^i_{\text{gravity}} + f^i_{\text{CoP}}$$

$$= R^i_{\text{sh},s_r} f^i_{s_r} + f^i_{\text{gravity}} + R^i_{\text{sh},c}$$

$$= M_{\text{sh}} \dddot{x}^i_{\text{sh},r}$$

$$= M_{\text{sh}} \frac{d}{dt} \left( R^i_{\text{sh},x_{\text{sh},r}} \right)$$

$$= M_{\text{sh}} \frac{d}{dt} \left( \frac{dR^i_{\text{sh},x_{\text{sh},r}}}{dt} + R^i_{\text{sh},\dot{x}_{\text{sh},r}} \right)$$

$$= M_{\text{sh}} \left( R^i_{\text{sh},S(\dot{\omega}_{\text{sh},r})x_{\text{sh},r}} + R^i_{\text{sh},S(\ddot{\omega}_{\text{sh},r})\dot{x}_{\text{sh},r}} + R^i_{\text{sh},S(\omega_{\text{sh},r})\dddot{x}_{\text{sh},r}} \right)$$

$$= M_{\text{sh}} \left( 2R^i_{\text{sh},S(\omega_{\text{sh},r})x_{\text{sh},r}} + R^i_{\text{sh},\dddot{x}_{\text{sh},r}} \right)$$

$$\Rightarrow f^i_{s_r} + (R^i_{\text{sh},R^i_{\text{sh},r}})^{-1} f^i_{\text{gravity}} + f^i_{\text{CoP}} = M_{\text{sh}} \left( 2S(\omega_{\text{sh},r})x_{\text{sh},r} + \dddot{x}_{\text{sh},r} \right)$$

Sum of moments:

$$\sum m^i_{\text{foot}} = R^i_{\text{sh},m^i_{\text{sh},r}} + (x^i_{\text{sh},r} + R^i_{\text{sh},x^i_{\text{sh},r}} - \dot{x}^i_{\text{sh},r} - R^i_{\text{sh},\dddot{x}_{\text{sh},r}}) \times R^i_{\text{sh},f^i_{s_r}}$$

$$+ (x^i_{\text{sh},r} + R^i_{\text{sh},x^i_{\text{sh},r}} - \dot{x}^i_{\text{sh},r} - R^i_{\text{sh},\dddot{x}_{\text{sh},r}}) \times R^i_{\text{sh},f^i_{\text{gravity}}}$$

$$+ R^i_{\text{sh},m^i_{\text{CoP}}} + (x^i_{\text{sh},r} + R^i_{\text{sh},x^i_{\text{sh},r}} - \dot{x}^i_{\text{sh},r} - R^i_{\text{sh},\dddot{x}_{\text{sh},r}}) \times R^i_{\text{sh},f^i_{\text{CoP}}}$$

$$= R^i_{\text{sh},m^i_{\text{sh},r}} + R^i_{\text{sh},m^i_{\text{CoP}}} + R^i_{\text{sh},f^i_{\text{sh},r}} \times \left( x^i_{\text{sh},r} + x^i_{\text{CoP}} \right) \times R^i_{\text{sh},f^i_{\text{sh},r}}$$

$$= R^i_{\text{sh},m^i_{\text{sh},r}} \frac{d}{dt} \left( R^i_{\text{sh},\omega_{\text{sh},r}} \right)$$

$$= R^i_{\text{sh},m^i_{\text{sh},r}} \frac{d}{dt} \left( \frac{dR^i_{\text{sh},\omega_{\text{sh},r}}}{dt} + R^i_{\text{sh},\dot{\omega}_{\text{sh},r}} \right)$$

$$= R^i_{\text{sh},m^i_{\text{sh},r}} \left( R^i_{\text{sh},S(\omega_{\text{sh},r})\dot{\omega}_{\text{sh},r}} + R^i_{\text{sh},\dddot{\omega}_{\text{sh},r}} \right)$$

$$= R^i_{\text{sh},m^i_{\text{sh},r}} \left( \omega_{\text{sh},r} + \dddot{x}_{\text{sh},r} \right)$$

$$\Rightarrow m^i_{s_r} + S(x^i_{s_r})f^i_{s_r} + m^i_{\text{CoP}} + S(x^i_{\text{CoP}})f^i_{\text{CoP}} = f^i_{\text{sh},r} \dddot{x}_{\text{sh},r}$$
Force work:

\[
W_{F,CoP} = \int_{0}^{1} \frac{d}{dt} \left( x_{shr}^i + R_{shr}^i x_{CoP,fix}^sh \right) \\
= \int_{0}^{1} \frac{d}{dt} \left( \dot{x}_{shr}^i + R_{shr}^i S(\omega_{shr}^i) x_{CoP,fix}^sh \right) \\
= \left( R_{shr}^i x_{CoP,\perp}^{sh} \right)^T \left( \dot{x}_{shr}^i + R_{shr}^i S(\omega_{shr}^i) x_{CoP,fix}^sh \right) \\
= R_{shr}^i x_{CoP,\perp}^{sh} \left( \dot{x}_{shr}^i + R_{shr}^i S(\omega_{shr}^i) x_{CoP,fix}^sh \right) \leq 0
\]  \quad (2.34)

Moment work:

\[
W_{M,CoP} = \left( R_{shr}^i m_{CoP}^sh \right)^T R_{shr}^i \omega_{CoP}^{sh} \\
= m_{CoP}^sh \omega_{shr}^{i,sh} \leq 0
\]  \quad (2.35)

Friction models:

\[
R_{f}^i e_{CoP,\perp}^c + R_{f}^i e_{CoP,\parallel}^c = R_{f}^i e_{CoP}^c \\
\Rightarrow e_{CoP,\perp}^c = \frac{\omega_{CoP}}{R_{f}^i} \parallel e_{CoP,\parallel}^c = \frac{\omega_{CoP}}{R_{f}^i} \\
\Rightarrow f_{CoP,\perp}^c = \frac{\omega_{CoP}}{R_{f}^i} \parallel f_{CoP,\parallel}^c = \frac{\omega_{CoP}}{R_{f}^i} \\
R_{f}^i e_{CoP,\perp}^c \times R_{f}^i e_{CoP,\parallel}^c = 0 \\
\Rightarrow f_{CoP,\perp}^c \times \dot{x}_{CoP,fix}^{sh} = 0 \\
R_{f}^i e_{CoP,\parallel}^c \times R_{f}^i e_{CoP,\parallel}^c = 0 \\
\Rightarrow m_{CoP}^c \times \omega_{CoP,fix}^{sh} = 0
\]  \quad (2.36)

Note that the component 2M_{shr} S(\omega_{shr}^i) x_{shr}^{i,sh} appearing as the result of Equation (2.32) is none other than the Coriolis force. Finally, the final equations for the foot are given by the following equations.

\[
f_{shr}^i + \left( R_{shr}^i R_{shr}^{i,gravity} \right)^{-1} f_{shr}^i = M_{shr} \left( 2 S(\omega_{shr}^i) \dot{x}_{shr}^{i,sh} + \dot{x}_{shr}^{i,sh} \right) \\
m_{shr}^i + S(\omega_{shr}^i) f_{shr}^i + m_{CoP}^i + S(\omega_{CoP}^i) f_{CoP}^i = f_{shr}^{i,sh} \\
\Rightarrow f_{shr}^i \omega_{shr}^{i,sh} \leq 0
\]  \quad (2.40a)

\[
m_{CoP}^i + S(\omega_{shr}^i) f_{shr}^i + m_{CoP}^i + S(\omega_{CoP}^i) f_{CoP}^i = f_{shr}^{i,sh} \\
\Rightarrow m_{CoP}^i \omega_{shr}^{i,sh} \leq 0
\]  \quad (2.40b)

\[
f_{shr}^i \omega_{shr}^{i,sh} \leq 0
\]  \quad (2.40c)

\[
f_{shr}^i \omega_{shr}^{i,sh} \leq 0
\]  \quad (2.40d)

\[
f_{shr}^i \omega_{shr}^{i,sh} \leq 0
\]  \quad (2.40e)

\[
f_{shr}^i \omega_{shr}^{i,sh} \leq 0
\]  \quad (2.40f)

\[
f_{shr}^i \omega_{shr}^{i,sh} \leq 0
\]  \quad (2.40g)

\[
f_{shr}^i \omega_{shr}^{i,sh} \leq 0
\]  \quad (2.40h)
The unknown variables were separated into two categories:

- **CoP:** \( x_{\text{CoP}}^{sh_r}, f_{\text{CoP}}^{sh_r}, m_{\text{CoP}}^{sh_r}, f_{\text{CoP}||}^{sh_r}, f_{\text{CoP}\perp}^{sh_r} \)
- **Foot dynamics:** \( \dot{x}_{sh_r}^{i}, \omega_{sh_r}^{i}, R_{sh_r}^{i}, \dot{\omega}_{sh_r}^{i}, \dot{\omega}_{sh_r}^{i} \)

When considering foot dynamics, then there are many more unknown variables than when compared to the static case. There are two possibilities in order to estimate the unknown foot dynamics variables:

1. **Base IMU**
   The foot dynamics could be estimated with respect to the base IMU. As the dynamics of the base can be estimated through integration, the dynamics of the foot can be computed using the joint angles and velocities. This method may be unreliable, as the estimation of the dynamics of the foot would be both dependent on the accuracy of measurements of the base IMU and the resolution of the joint angles.

2. **Foot IMU**
   The foot dynamics could be estimated more precisely by adding an IMU on each foot. From this IMU, the dynamics could actually be estimated through direct measurement and integration. This method has the advantage over the previous one of not being dependent on the joint angles.

In both cases, only IMU sensors are proposed in order to estimate the dynamics. This method requires to integrate the measurements, which is of course prone to drift due to measurement noise. Also note that the two methods are not exclusive, and could both be used together in order to profit from the redundancy and obtain a more reliable estimation.

In the case of this project, it was decided that no hardware modification or addition would be applied to the robot. As such, the first case **Base IMU** was implemented. The foot dynamics can then be related to the Base dynamics when assuming rigid bodies and using the joint angles. If the foot dynamics are estimated, then the unknown CoP variables are expected to have the following structure:

\[
\begin{bmatrix}
    f_{\text{CoP}}^{sh_r} \\
    m_{\text{CoP}}^{sh_r} \\
    x_{\text{CoP}}^{sh_r}
\end{bmatrix} =
    \begin{bmatrix}
        f_1 \\
        f_2 \\
        f_3
    \end{bmatrix},
\begin{bmatrix}
    \tau_1 \\
    \tau_2 \\
    \tau_3
\end{bmatrix} =
    \begin{bmatrix}
        \frac{d}{dt} x_1 \\
        \frac{d}{dt} x_2 \\
        0
\end{bmatrix} \text{ (2.41)}
\]

It is important to note that Equations (2.40g) is used to set the CoP friction and velocity collinear, which could limit drift. Fusing the velocity of the foot with the force/torque sensors may be expected to correct the estimation as the movement of the feet would be limited to a certain direction measure by the force/torque sensors when sliding. Furthermore, it is expected that the same equation from (2.40) could be applied when the feet are not in contact with the ground. Indeed, \( f_{\text{CoP}||}^{sh_r} \) would then be expected to be be equal to zero, allowing the foot to move freely in the directions obtained from the base IMU and the joint angles. Furthermore, if both feet were off the ground, then the dynamics of the robot would purely rely on the base IMU. Finally, the feet can be related to the base using the following equations:

**Position:**

\[
x_{sh_r}^{i} = x_{sh_r}^{i} + R_{sh_r}^{i} x_{sh_r}^{b} \text{ (2.42)}
\]

**Velocity:**

\[
\begin{align*}
\dot{x}_{sh_r}^{i} &= \frac{d}{dt} (x_{sh_r}^{i} + R_{sh_r}^{i} x_{sh_r}^{b}) \\
&= \frac{dx_{sh_r}^{i}}{dt} + \frac{dR_{sh_r}^{i}}{dt} x_{sh_r}^{b} + R_{sh_r}^{i} \frac{dx_{sh_r}^{b}}{dt} \\
&= \dot{x}_{sh_r}^{i} + R_{sh_r}^{i} S(\omega_{sh_r}^{b}) x_{sh_r}^{b} + R_{sh_r}^{i} \dot{x}_{sh_r}^{b} \text{ (2.43)}
\end{align*}
\]
Acceleration:

\[
\ddot{x}_{shr}^i = \frac{d}{dt} \left( \dot{x}_{b}^i + R_1^b S(\omega_b^{i,b}) x_{shr}^b + R_1^b \dot{x}_{shr}^b \right)
\]

\[
= \frac{d\dot{x}_{b}^i}{dt} + R_1^b S(\omega_b^{i,b}) \dot{x}_{shr}^b + \frac{dR_1^b}{dt} dS(\omega_b^{i,b}) x_{shr}^b + R_1^b \ddot{x}_{shr}^b + \frac{dR_1^b}{dt} \dot{x}_{shr}^b
\]

\[
= \ddot{x}_{b}^i + R_1^b S(\omega_b^{i,b}) \dot{x}_{shr}^b + R_1^b \ddot{x}_{shr}^b
\]

\[
= \ddot{x}_{b}^i + R_1^b S(\omega_b^{i,b}) \dot{x}_{shr}^b + R_1^b \ddot{x}_{shr}^b
\]

(2.44)

Orientation:

\[
R_1^i_{shr} = R_1^b R_{shr}^b
\]

(2.45)

Angular velocity:

\[
\omega_{shr}^{i,b} = R_1^i \omega_{shr}^b = R_1^i R_{shr}^b \omega_{shr}^b = R_1^i R_{shr}^b (\omega_b^i + R_1^b \omega_{shr}^b)
\]

(2.46)

Angular acceleration:

\[
\ddot{\omega}_{shr}^{i,b} = \frac{d}{dt} \left( R_1^i R_{shr}^b (\omega_b^i + R_1^b \omega_{shr}^b) \right)
\]

\[
= \frac{dR_1^i}{dt} R_{shr}^b (\omega_b^i + R_1^b \omega_{shr}^b) + \frac{dR_1^b}{dt} R_{shr}^b (\omega_b^i + R_1^b \omega_{shr}^b)
\]

\[
= \ddot{\omega}_{shr}^{i,b} + R_1^b \omega_{shr}^b \frac{d}{dt} (\omega_b^i + R_1^b \omega_{shr}^b)
\]

\[
= R_1^i S(\omega_b^{i,b}) R_{shr}^b (\omega_b^i + R_1^b \omega_{shr}^b) + R_1^i R_{shr}^b S(\omega_{shr}^{b,shr}) (\omega_b^i + R_1^b \omega_{shr}^b)
\]

\[
+ \frac{dR_1^b}{dt} \omega_{shr}^b + \frac{dR_1^b}{dt} \omega_{shr}^b + \frac{d\omega_{shr}^b}{dt}
\]

\[
= (R_1^i S(\omega_b^{i,b}) R_{shr}^b + R_1^i R_{shr}^b S(\omega_{shr}^{b,shr})) (\omega_b^i + R_1^b \omega_{shr}^b)
\]

\[
+ R_1^b R_{shr}^b (\omega_b^i + R_1^b \omega_{shr}^b + R_1^b \omega_{shr}^b)
\]
2.6 Prediction Model

In the Kalman filter, a prediction model is used. A prediction model must allow to simulate as accurately as possible the future state of the system knowing a current state and the inputs into the model. For a robot, the system is commonly assumed to be governed by the following equations:

\[
M(q)\ddot{q} + h(q, \dot{q}) = S\tau + J_c^T(q)\lambda
\]

\[
\dot{x}_C = J'_c\dot{q} + J'_{cC}\dot{q}
\]

This was typically implemented by [5], where they explain this structure as follows:

- \( q = [p_x, p_q, \theta_J]^T \) is the vector of base positions, orientations and joint angles,
- \( \dot{q} = [p_v, p_\omega, \dot{\theta}_J]^T \) is the vector of base velocity, angular velocity and joint velocities,
- \( M(q) \) is the inertia matrix,
- \( h(q, \dot{q}) \) is the vector sum of the vector sum for Coriolis, centripetal and gravitational forces,
- \( S = [0; I]^T \) is the selection matrix with zero entries corresponding to the base variables and the identity matrix corresponding to the joint variables,
- \( \tau \) is the vector for actuating joint torques,
- \( J_c(q) \) is the Jacobian matrix and \( J'_c(q, \dot{q}) \) is its derivative at contact points, and
- \( f \) is the vector of external forces at contact points in the world frame.

Note that the problem with using a prediction is that the model must be very accurate. Particularly in the case of this project, where there are no sensors used for directly measuring the location of the robot in the global frame. As such, an inaccurate model would cause error in the state estimation itself. Moreover, a model which does not predict external disturbance would also cause these types of errors, as explained by [4] and discussed in Section 2.2. As such, a prediction model was not used in this project. However, if a prediction model was required, constrained optimisation allows to fuse it directly to the state estimation obtained with measurements. As such, the state estimation algorithms designed during this project and presented in Section 2.7 could take model prediction into account, which was done by [5] when using quadratic programming.

2.7 State estimation

As proposed previously, nonlinear constrained optimisation was to be used in order to compute the state estimation. This approach has the advantage of being computationally fast for large scale problems and can solve problems given constraints. Furthermore, the state is expected to be close to the result obtained in the previous iteration. Thus this information can be used in order to initiate the convergence from a close point, known as a warm start. This also helps to avoid the result being biased by local minima caused by the nature of the non-linearity. This also allows to limit the number of iterations permitted for the optimisation to solve the problem, which is useful for online computation on the robot. In order to deal with the complexity of the implementation, the approach considered was to design multiple algorithms iteratively. Indeed, any mistake in the implementation will often result in widely incorrect results. Finding the cause of an error is then quite challenging. As such, the static algorithm was first reimplemented and modified before implementing the dynamic contact.

2.7.1 Static contact

As implemented in [1], the static optimisation was set with the following problem:
min_{\dot{y}, x_{\text{CoP}}^c, x_{\text{CoP}}^l} V_{Q_{s_1}}(\delta_r) + V_{Q_{s_2}}(\delta_l) + V_{Q_{y}}(\dot{y} - \dot{y}_{\text{ref}}) + V_{Q_{m_{\text{CoP}}}}(m_{\text{CoP}}^c) + V_{Q_{m_{\text{CoP}}}}(m_{\text{CoP}}^l) \quad (2.50a)

s.t.

\begin{align*}
    m_{\text{CoP}}^c &= A_1 x_{\text{CoP}}^c + B_1 \\
    m_{\text{CoP}}^l &= A_2 x_{\text{CoP}}^l + B_2 \\
    -\dot{y} + \delta_r &= A_3 x_{\text{CoP}}^c + B_3 \\
    -\dot{y} + \delta_l &= A_4 x_{\text{CoP}}^l + B_4 \\
    x_{\text{CoP}}^c, x_{\text{CoP}}^l &\in \text{support polygon surface} \quad (2.50b, c, d, e, f)
\end{align*}

where $V_Q(x) = x^T Q x$ and:

\begin{align*}
    \dot{y}_{\text{ref}} &= \dot{y}^- + \dot{x}_{\text{imu}}^b dt \\
    A_1 &= S(f_{c_r}^c + f_{c_l}^c) \\
    B_1 &= m_{c_r}^c - S(f_{g}^c)(x_{c_r}^b - x_{c_r}^r) - S(f_{s_1}^r)(x_{s_1}^b - x_{s_1}^r) \\
    A_2 &= S(f_{c_i}^i + f_{c_i}^l) \\
    B_2 &= m_{s_1}^c - S(f_{g}^c)(x_{s_1}^l - x_{c_i}^l) - S(f_{s_1}^r)(x_{s_1}^b - x_{s_1}^r) \\
    A_3 &= R_{c_r}^b S(\omega_{c_r}^b, c_r) + S(\omega_{i,b}^b) R_{c_r}^b \\
    B_3 &= \dot{x}_{c_r} + S(\omega_{i,b}^b) x_{c_r} \\
    A_4 &= R_{c_i}^b S(\omega_{i,b}^b, c_i) + S(\omega_{i,b}^b) R_{c_i}^b \\
    B_4 &= \dot{x}_{c_i} + S(\omega_{i,b}^b) x_{c_i} \\
    \omega_{i,b}^b &= \omega_{\text{imu}}^b \\
    f_{c_r}^c &= (R_{c_r}^b R_{c_i}^b)^{-1} g^l M_{sh} \\
    f_{g}^c &= (R_{c_r}^b R_{c_i}^b)^{-1} g^l M_{sh} \\
\end{align*}

$\dot{y}^-$ is obtained from the previous iteration. The first two constraints (2.50b) and (2.50c) are based on Equations (2.16) and (2.17). where (2.50b) corresponds to the right foot, and (2.50c) corresponds to the left foot. They directly link the moments $m_{\text{CoP}}^c$ and $m_{\text{CoP}}^l$ reaction of the ground directly to the centres of pressure $x_{\text{CoP}}^c$ and $x_{\text{CoP}}^l$ for each foot. The following 2 constraints (2.50d) and (2.50e) link the centre of pressure of each foot to the velocity of the base. Note that the moments $m_{\text{CoP}}^c$ and $m_{\text{CoP}}^l$ are only minimised along the first two axis of $c_r$ in the objective function by designing the cost accordingly, i.e. the vectors of the CoP moments are perpendicular to the contact surface as discussed in Section 2.5.3.

This algorithm based on static contact was written as the first part of a state estimation. The second part was written by directly solving the position and orientation states of the base. This could have also been done by adding the relations into the optimisation algorithm. This may lead to the advantage of having more redundancy as the error of the joint angles could also be considered, but at the cost of higher complexity. In order to solve the dynamic contact, it was of interest to know if it was possible to implement all physical and geometrical relations as the optimisation constraints. As such, the next step was to implement the state estimation as follows. This implementation would be referred to as the intermediate static state estimation.
\[
\begin{align*}
&\min_{\text{state}} V_{Q_s}(\delta_s) + V_{Q^s_{fr}}(\delta_{fr}) + V_{Q^s_{fl}}(\delta_{fl}) + V_{Q^s_{mr}}(\delta_{mr}) + V_{Q^s_{ml}}(\delta_{ml}) + \\
&V_{Q^s_{x_l}}(\delta_{x_l}) + V_{Q^s_{x_r}}(\delta_{x_r}) + V_{Q^s_{y_l}}(\delta_{y_l}) + V_{Q^s_{y_r}}(\delta_{y_r}) + V_{Q^s_{\omega_l}}(\delta_{\omega_l}) + V_{Q^s_{\omega_r}}(\delta_{\omega_r})
\end{align*}
\]
\[\text{s.t.}\]
\[
\begin{align*}
m_{C_{\text{CoP}}}^c &= A_1x_{C_{\text{CoP}}}^c + B_1 \\
m_{C_{\text{CoP}}}^c &= A_2x_{C_{\text{CoP}}}^c + B_2 \\
-\dot{y} + \delta_r &= A_3x_{C_{\text{CoP}}}^c + B_3 \\
-\dot{y} + \delta_l &= A_4x_{C_{\text{CoP}}}^c + B_4 \\
y + R_b^i x_{sh_r}^i + \delta_{x_l} &= x_{sh_r,\text{fix}}^i \\
y + R_b^i x_{sh_r}^i + \delta_{x_r} &= x_{sh_r,\text{fix}}^i \\
y &= y^r + R_b^r \dot{y}_dl + \dot{\delta}_y \\
\dot{y} &= \dot{y}_{\text{imu}} + \delta_y \\
\dot{y}_{\text{imu}} &= y^r - x_{\text{imu}}^i \dot{y}dt + \dot{\delta}_y \\
w &= \omega_{\text{imu}} + \delta_\omega \\
w + \omega_{\text{imu},\text{fix}} + \delta_\omega &= 0 \\
w + \omega_{\text{imu},\text{fix}} + \delta_\omega &= 0 \\
y + R_b^i x_{sh_r}^i - x_{\text{imu}}^i \delta_x &= 0 \\
y + R_b^i x_{sh_r}^i - x_{\text{imu}}^i \delta_x &= 0 \\
y + R_b^i S(w + \delta_\omega) x_{sh_r}^i + R_b^i \dot{x}_{sh_r}^i &= 0 \\
y + R_b^i S(w + \delta_\omega) x_{sh_r}^i + R_b^i \dot{x}_{sh_r}^i + \delta_\omega &= 0 \\
x_{\text{CoP}}^c, x_{\text{CoP}}^c &\in \text{support polygon surface}
\end{align*}
\]

where $A_1, A_2, A_3, A_4, B_1, B_2, B_3$ and $B_4$ are the same matrices as described in Equations (2.51).

The advantage of this algorithm is that it would also compute the position of and angular velocity of the base by fusing the IMU data with the feet. The results obtained using this approach was expected to yield the same results as those obtained by the two steps implemented by [1]. The final iteration for the static contact would be of the following form to test an approach where the physical and geometrical relations are directly provided. This implementation would be referred to as the full static state estimation.

\[
\begin{align*}
&\min_{\text{state}} V_{Q_s}(\delta_s) + V_{Q^s_{fr}}(\delta_{fr}) + V_{Q^s_{fl}}(\delta_{fl}) + V_{Q^s_{mr}}(\delta_{mr}) + V_{Q^s_{ml}}(\delta_{ml}) + \\
&V_{Q^s_{x_l}}(\delta_{x_l}) + V_{Q^s_{x_r}}(\delta_{x_r}) + V_{Q^s_{y_l}}(\delta_{y_l}) + V_{Q^s_{y_r}}(\delta_{y_r}) + V_{Q^s_{\omega_l}}(\delta_{\omega_l}) + V_{Q^s_{\omega_r}}(\delta_{\omega_r})
\end{align*}
\]
\[\text{s.t.}\]
\[
\begin{align*}
r_{sh} + \delta_r &= (R_b^i x_{sh_r}^i)^{-1} r_{\text{gravity}} + x_{\text{CoP}}^c \\
r_{sh} + \delta_l &= (R_b^i x_{sh_l}^i)^{-1} r_{\text{gravity}} + x_{\text{CoP}}^c \\
m_{C_{\text{CoP}}}^c &= m_{C_{\text{CoP}}}^c + m_{C_{\text{CoP}}}^c + S(x_{\text{CoP}}^c) x_{sh}^c + S(x_{\text{CoP}}^c) x_{sh}^c \\
m_{C_{\text{CoP}}}^c &= m_{C_{\text{CoP}}}^c + m_{C_{\text{CoP}}}^c + S(x_{\text{CoP}}^c) x_{sh}^c + S(x_{\text{CoP}}^c) x_{sh}^c \\
y - y^r - R_b^r \dot{y}_dl + \dot{\delta}_y &= 0 \\
\dot{y} - \dot{y}_{\text{imu}} + \delta_y &= 0 \\
\dot{y}_{\text{imu}} - y^r - x_{\text{imu}}^i \dot{y}dt + \dot{\delta}_y &= 0 \\
w - \omega_{\text{imu}}^b + \delta_\omega &= 0 \\
w - \omega_{\text{imu}}^b + \delta_\omega &= 0 \\
w + \omega_{\text{imu}}^b + \delta_\omega &= 0 \\
w + \omega_{\text{imu}}^b + \delta_\omega &= 0 \\
y + R_b^i x_{sh_r}^i - x_{\text{imu}}^i \delta_x &= 0 \\
y + R_b^i x_{sh_r}^i - x_{\text{imu}}^i \delta_x &= 0 \\
y + R_b^i S(w + \delta_\omega) x_{sh_r}^i + R_b^i \dot{x}_{sh_r}^i &= 0 \\
y + R_b^i S(w + \delta_\omega) x_{sh_r}^i + R_b^i \dot{x}_{sh_r}^i + \delta_\omega &= 0 \\
x_{\text{CoP}}^c, x_{\text{CoP}}^c &\in \text{support polygon surface}
\end{align*}
\]
The advantage of this method is that the constraints are directly given as each of the physical and geometrical relations. This was possible as the library used can solve the optimisation algorithm with nonlinear constraints. In order to be closer to the implementation of the Kalman filter, it would have been interesting to add an error to each sensor as a slack variable. This would have then allowed to assign the cost of each of these errors as the covariance matrix of the measurements errors. Unfortunately, this approach was not successfully achieved, as SNOPT would lose its ability to converge to a solution. This was either caused by an implementation error or due to the difficulty to solve the highly complex nonlinear problem. Typically, these kind of errors were observed to happen when the slack variables were written inside a skew-symmetric matrix. As such, the costs were inserted as an addition to each equation. It is still possible to imagine reassigning the value of each cost for these slack variables as a function of the covariance matrices of the equation itself, which would be expected to yield the same results.
2.7.2 Dynamic contact

For the dynamic contact, the equations from (2.40) can be put together inside a constrained optimisation in order to obtain the estimate of the state from the measurements of sensors as follows.

\[
\begin{align*}
    \min_{\text{state}} & \quad V_{Q_{l_{1r}}} (\delta_{1r}) + V_{Q_{r_{1l}}} (\delta_{1l}) + V_{Q_{l_{2r}}} (\delta_{2r}) + V_{Q_{r_{2l}}} (\delta_{2l}) + \\
    & + V_{Q_{l_{3r}}} (\delta_{3r}) + V_{Q_{r_{3l}}} (\delta_{3l}) + V_{Q_{l_{4r}}} (\delta_{4r}) + V_{Q_{r_{4l}}} (\delta_{4l}) + \\
    & + V_{Q_{l_{5r}}} (\delta_{5r}) + V_{Q_{r_{5l}}} (\delta_{5l}) + V_{Q_{l_{6r}}} (\delta_{6r}) + V_{Q_{r_{6l}}} (\delta_{6l}) + \\
    & + V_{Q_{l_{7r}}} (\delta_{7r}) + V_{Q_{r_{7l}}} (\delta_{7l}) + V_{Q_{l_{8r}}} (\delta_{8r}) + V_{Q_{r_{8l}}} (\delta_{8l}) + \\
    & + V_{Q_{l_{9r}}} (\delta_{9r}) + V_{Q_{r_{9l}}} (\delta_{9l}) + V_{Q_{l_{10r}}} (\delta_{10r}) + V_{Q_{r_{10l}}} (\delta_{10l}) \tag{2.54a}
\end{align*}
\]

\[
\begin{align*}
    f^\text{sh}_{s_t} + (R^b_{\text{sh}, b_{\text{sh}}})^{-1} f^i_{\text{gravity}} + f^\text{sh}_{\text{CoP}} - M_{\text{sh}, r} \left( 2S(\omega^0_{\text{sh}, r}) x^0_{\text{sh}, r} + x^0_{\text{sh}, r} \right) + \delta_{1r} = 0 \tag{2.54b}
\end{align*}
\]

\[
\begin{align*}
    f^\text{sh}_{s_l} + (R^b_{\text{sh}, b_{\text{sh}}})^{-1} f^i_{\text{gravity}} + f^\text{sh}_{\text{CoP}} - M_{\text{sh}, l} \left( 2S(\omega^0_{\text{sh}, l}) x^0_{\text{sh}, l} + x^0_{\text{sh}, l} \right) + \delta_{1l} = 0 \tag{2.54c}
\end{align*}
\]

\[
\begin{align*}
    m^\text{sh}_{s_t} + S(x^0_{\text{sh}, r}) f^\text{sh}_{s_t} + m^\text{sh}_{\text{CoP}} + S(x^0_{\text{CoP}}) f^\text{sh}_{\text{CoP}} - I^0_{\text{sh}, r} x^0_{\text{sh}, r} + \delta_{2r} = 0 \tag{2.54d}
\end{align*}
\]

\[
\begin{align*}
    m^\text{sh}_{s_l} + S(x^0_{\text{sh}, l}) f^\text{sh}_{s_l} + m^\text{sh}_{\text{CoP}} + S(x^0_{\text{CoP}}) f^\text{sh}_{\text{CoP}} - I^0_{\text{sh}, l} x^0_{\text{sh}, l} + \delta_{2l} = 0 \tag{2.54e}
\end{align*}
\]

\[
\begin{align*}
    f^\text{sh}_{\text{CoP}}^T (x^0_{\text{sh}, r} + S(\omega^0_{\text{sh}, r}) x^0_{\text{CoP}, \text{fix}}) + \delta_{3r} \leq 0 \tag{2.54f}
\end{align*}
\]

\[
\begin{align*}
    f^\text{sh}_{\text{CoP}}^T (x^0_{\text{sh}, l} + S(\omega^0_{\text{sh}, l}) x^0_{\text{CoP}, \text{fix}}) + \delta_{3l} \leq 0 \tag{2.54g}
\end{align*}
\]

\[
\begin{align*}
    m^\text{sh}_{\text{CoP}} \omega^0_{\text{sh}, r} + \delta_{4r} \leq 0 \tag{2.54h}
\end{align*}
\]

\[
\begin{align*}
    m^\text{sh}_{\text{CoP}} \omega^0_{\text{sh}, l} + \delta_{4l} \leq 0 \tag{2.54i}
\end{align*}
\]

\[
\begin{align*}
    f^\text{sh}_{\text{CoP}, \perp} + f^\text{sh}_{\text{CoP}, \parallel} - f^\text{sh}_{\text{CoP}} + \delta_{5r} = 0 \tag{2.54j}
\end{align*}
\]

\[
\begin{align*}
    f^\text{sh}_{\text{CoP}, \perp} + f^\text{sh}_{\text{CoP}, \parallel} - f^\text{sh}_{\text{CoP}} + \delta_{5l} = 0 \tag{2.54k}
\end{align*}
\]

\[
\begin{align*}
    f^\text{sh}_{\text{CoP}, \perp}^T f^\text{sh}_{\text{CoP}, \parallel} + \delta_{6r} = 0 \tag{2.54l}
\end{align*}
\]

\[
\begin{align*}
    f^\text{sh}_{\text{CoP}, \perp}^T f^\text{sh}_{\text{CoP}, \parallel} + \delta_{6l} = 0 \tag{2.54m}
\end{align*}
\]

\[
\begin{align*}
    f^\text{sh}_{\text{CoP}, \parallel} x^0_{\text{CoP}, \text{fix}} + \delta_{7r} = 0 \tag{2.54n}
\end{align*}
\]

\[
\begin{align*}
    f^\text{sh}_{\text{CoP}, \parallel} x^0_{\text{CoP}, \text{fix}} + \delta_{7l} = 0 \tag{2.54o}
\end{align*}
\]

\[
\begin{align*}
    m^\text{sh}_{\text{CoP}} \omega^0_{\text{CoP}, \text{fix}} + \delta_{8r} = 0 \tag{2.54p}
\end{align*}
\]

\[
\begin{align*}
    m^\text{sh}_{\text{CoP}} \omega^0_{\text{CoP}, \text{fix}} + \delta_{8l} = 0 \tag{2.54q}
\end{align*}
\]

\[
\begin{align*}
    y - y^r - R^b_{y_{\text{imu}}} \\ y - y^r - \dot{x}_{\text{imu}} + \delta_{y} = 0 \tag{2.54r}
\end{align*}
\]

\[
\begin{align*}
    \dot{y}_{\text{imu}} - \dot{y} - \dot{x}^b_{\text{imu}} dl + \delta_{y} = 0 \tag{2.54s}
\end{align*}
\]

\[
\begin{align*}
    w - \omega^b_{\text{imu}} + \delta_{\omega} = 0 \tag{2.54t}
\end{align*}
\]

\[
\begin{align*}
    w + \omega^b_{\text{imu}} + \delta_{\omega} = 0 \tag{2.54u}
\end{align*}
\]

\[
\begin{align*}
    w + \omega^b_{\text{imu}} + \delta_{\omega} = 0 \tag{2.54v}
\end{align*}
\]

\[
\begin{align*}
    x^0_{\text{CoP}} \cdot x^0_{\text{CoP}} \in \text{support polygon surface} \tag{2.54w}
\end{align*}
\]
2.8 Implementation

2.8.1 Optimisation

The first algorithm to implement was the first step of the state estimation designed in [1]. The original implementation was provided, meaning that the results could be compared to verify a correct implementation and check that the libraries would work as expected. This was a necessary step in order to familiarise with the implementation of constrained optimisation problems. The first step of this state estimation was designed to consider static contact and was originally implemented using the CVXGEN optimisation library. This algorithm was first reimplemented using the Scipy package in Python using the SLSQP method [28]. Scipy offered the possibility of easily implementing and testing the algorithm, which allowed to familiarise with the general implementation of optimisation problems. Next, the first step was again reimplemented with the SNOPT package. SNOPT had the advantage of being capable of computing accurate results much faster as its implementation is coded directly in Fortran, making SNOPT a more suitable choice in the case of the project. In addition, this problem would yield a total amount of 27 variables to compute using the objective function with 24 constraints, i.e. when the constraints are written as scalar equations. This would result in a sparse Jacobian of 93 non-zero elements.

Writing the constrained optimisation was found to be quite challenging due to the complexity encountered. Indeed, the results could only be compared once the full algorithm would be implemented. The origin of any mistake would be challenging to find. As such, a code generator script using the Sympy package with Python was written to generate the C++ code that interfaces to SNOPT. Furthermore, the advantage is that this script could also automatically compute the Jacobian of the objective function, as well as for the constraints. If another library were to be used, the computation of the Hessian could also be expected to be done using such a script, as well as generating the corresponding code for it. Further information about the code generation for the SNOPT library can be found in Appendix C.2.

The implementation was then completed iteratively by modifying the constraints one by one, allowing to check that the implementation was done correctly. The results would be compared to the state estimation designed by [1], which will be referred to as the adaptive odometry. Table 2 show some statistics for the different state estimations implementations.

<table>
<thead>
<tr>
<th>State estimation</th>
<th>nX</th>
<th>neF</th>
<th>neG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive odometry - step 1</td>
<td>27</td>
<td>25</td>
<td>93</td>
</tr>
<tr>
<td>Intermediate static</td>
<td>57</td>
<td>49</td>
<td>168</td>
</tr>
<tr>
<td>Full static</td>
<td>72</td>
<td>43</td>
<td>189</td>
</tr>
<tr>
<td>Dynamic</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
</tbody>
</table>

*Table 2: State estimation implementation statistics. nX corresponds to the number of variables that must be solved in the optimisation algorithm. neF corresponds to the number of functions used for the optimisation, which correspond to the objective function plus the number of constraints written as scalar equations. neG corresponds to the number of non-zero elements of the sparse Jacobian. Note that the number of equations in the full static is smaller than the intermediate state estimation as the CoP position constraints were directly implemented as state constraints instead.*

2.8.2 GUI

By simply printing the results, it would be too complicated to develop the state estimation due to the high dimensionality of the results. In order to visualise the results and interact with the robot during the execution of the controller, a GUI was implemented which would be launched each time the simulation scenario was initiated in Webots. It was completed by first embedding the Python interpreter into the C++ code as first mentioned in Section 2.4.2. The code written for Webots in C++ could then call the Python code, allowing it to exchange data. This method was also used in order to implement the state estimation with the Scipy package.

The GUI itself was run using multiprocessing, where a new process is spawned in order to avoid slowing down the webots threads. As such, an inter-process communication was set between the GUI and Webots using process pipes. The data could then be passed using these pipes, which would avoid needing to synchronise the processes. In order to avoid having the GUI constantly working on checking for new data
in the pipe, the check would be limited to about 40 times a second, allowing to plot the data smoothly. The Webots code would simulate a state estimation iteration once every 4 milliseconds in simulation time, corresponding to 250 iterations per second is simulating real-time. However, the simulation would typically run between 40-80 % of this speed, which would still run faster than the GUI. As such, the GUI would read all data from the pipe every 40 milliseconds, and pause the rest of the time. An illustration of this GUI can be found in Figures 2.7 for the control and 2.8 for the visualisation of the data. The switch between these two modes could be done using the tabs at the top. On a side-note, the plots obtained in the 'Plots' tab was implemented with PyQtGraph, which also provides certain tools, such as computing the FFT of the plotted signal. These kind of visualisation tools may be useful when directly interfacing the real coman robot to Webots, as the the noise model could be studied during testing. Finally, the plots could be published from the GUI using the Matplotlib package. An example is illustrated in Figure 2.9. These plots could also be saved directly from this interface, which was used to generate the results shown in Section 3.

![Figure 2.7: GUI - Control of the coman robot. In particular, the parameters to control for this project were the costs for the different algorithm implementation. Note that this control interface would have been useful when interfacing the robot to Webots, as the costs could then be configured online. However, since no noise model was implemented in the simulation, this control panel was not used.](image-url)
Figure 2.8: GUI - Visualisation of data using plots. From this window, the data sent from Webots could be plotted. This was of great assistance for debugging whenever undesired results would be obtained. Note that the checkbox at the top could be used to activate and deactivate the plotting. There are then two combo boxes under this checkbox. The first combo box allows to choose the component to plot, which includes sensors or state estimation results. The second combo box is used in the case that a state estimation was chosen, and allows to choose the state to plot. In this plot, the gyroscope data is plotted along time after the initialisation of the scenario. One may observe the transition of the robot from its initialisation to its permanent trajectory regime set by the controller. At the bottom of the window, a "Generate plot" button can be found, which allows to generate the plot observed with the Matplotlib package. Another button was also present in order to save the plot according to the options set in the editable text boxes.

Figure 2.9: Example of plots that would appear when pressing the "Generate plot" button from the "Plots" tab of the GUI. In this case, sensor data is shown for the static scenario.
3 Results

3.1 IMU integration

First, a simple integration was implemented in order to show the usefulness of considering static contact in order to avoid drift. Indeed, one may observe in Figure 3.1 that velocity would not have an average of zero along all axis. Furthermore, the position can be seen to quickly drift over the course of ten seconds when compared to the supervisor. Assuming static contact allows to cancel this drift over time.

![Figure 3.1: Integration of IMU. The position is also compared to the supervisor, where $x_{se}$, $y_{se}$ and $y_{se}$ correspond to the IMU integration in full lines, and $x_{sv}$, $y_{sv}$ and $y_{sv}$ correspond to the Webots supervisor in dashed lines, i.e. the "real" position of the robot in the world frame.](image)

3.2 Static contact scenario

In order to verify the results that were obtained during the project for the different state estimation algorithms, they were compared to the adaptive odometry implemented by [1]. This would work for the static simulation scenario. A Webots supervisor was also implemented which would only yield the position of the base. However, note that the position of the base would still be a good way to check results, as it would both depend on the velocities estimated, and the orientation of the robot to match the adaptive odometry.

3.2.1 Step 1

The first step of the adaptive odometry was reimplemented with both Scipy and SNOPT. The results are shown in Figures 3.2, 3.3 and 3.4, which provide the estimation of the velocity of the base in its local frame, the position of the CoP in the surface contact frame for the right foot and the position of the CoP for the left foot respectively.

![Figure 3.2: Comparison of estimated velocities](image)
3.2.2 Intermediate static state estimation

Next, the intermediate state estimation was implemented. The results are shown in Figures 3.5, 3.6, 3.7, 3.8 and 3.9. All the results show that this approach perfectly correlates with the adaptive odometry, meaning that it was implemented successfully.
Figure 3.6: Comparison of the estimated base velocity between the adaptive odometry and the intermediate static state estimation

Figure 3.7: Comparison of estimated angular velocity between the adaptive odometry and the intermediate static state estimation

Figure 3.8: Comparison of estimated right foot CoP position between the adaptive odometry and the intermediate static state estimation
3.2.3 Addition of noise to the intermediate state estimation

As the presence of noise is expected to be the main challenge, an objective was to test how the algorithm reacts to this addition. As such, a simulation of the noise was implemented using a normal distribution model for the IMU and the force/torque sensors. This would allow to have a basic idea of the results that may be obtained, although this is only an approximation of reality. As no noise or resolution of the encoders were modelled, only the states affected by the noise are presented. The comparison of the sensor with and without noise are presented in Figures 3.10, 3.11, 3.12, 3.13, 3.14, 3.15 and 3.16. The comparison of the estimated states are then illustrated in Figures 3.17, 3.18 and 3.19. The values of the standard deviations of the noise given to the sensors and computed for the states can be found in table 3. One may observe that the addition of noise has an important influence on the state estimation. In order to yield the best estimate, it is critical to correctly dimension the objective function costs.

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Noise STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer [m/s²]</td>
<td>0.02 g</td>
</tr>
<tr>
<td>Gyroscope [rad/s]</td>
<td>0.1</td>
</tr>
<tr>
<td>Magnetometer [–]</td>
<td>0.01</td>
</tr>
<tr>
<td>Force [N]</td>
<td>1</td>
</tr>
<tr>
<td>Moment [Nm]</td>
<td>0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Estimation</th>
<th>Intermediate state estimation - Adaptive odometry STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base velocity - x [m/s]</td>
<td>1.922 · 10⁻³</td>
</tr>
<tr>
<td>Base velocity - y [m/s]</td>
<td>2.037 · 10⁻³</td>
</tr>
<tr>
<td>Base velocity - z [m/s]</td>
<td>4.520 · 10⁻²</td>
</tr>
<tr>
<td>Right foot CoP position - x [m]</td>
<td>1.066 · 10⁻⁴</td>
</tr>
<tr>
<td>Right foot CoP position - y [m]</td>
<td>1.035 · 10⁻⁴</td>
</tr>
<tr>
<td>Right foot CoP position - z [m]</td>
<td>5.191 · 10⁻²⁰</td>
</tr>
<tr>
<td>Left foot CoP position - x [m]</td>
<td>1.106 · 10⁻⁴</td>
</tr>
<tr>
<td>Left foot CoP position - y [m]</td>
<td>1.321 · 10⁻⁴</td>
</tr>
<tr>
<td>Left foot CoP position - z [m]</td>
<td>1.929 · 10⁻¹⁵</td>
</tr>
</tbody>
</table>

Table 3: Standard deviations of the noised sensors and the estimated states.
Figure 3.10: Implementation of noise for the accelerometer

(a) Without noise

(b) With noise

Figure 3.11: Implementation of noise for the gyroscope

(a) Without noise

(b) With noise

Figure 3.12: Implementation of noise for the magnetometer

(a) Without noise

(b) With noise
Figure 3.13: Implementation of noise for the right foot force sensor

Figure 3.14: Implementation of noise for the left foot force sensor

Figure 3.15: Implementation of noise for the right foot torque sensor
(a) Without noise  
(b) With noise

*Figure 3.16: Implementation of noise for the left foot torque sensor*

(a) Adaptive odometry without noise  
(b) Intermediate state estimation with noise

*Figure 3.17: Estimation of the base velocity*

(a) Adaptive odometry without noise  
(b) Intermediate state estimation with noise

*Figure 3.18: Estimation of the right foot CoP position*
3.2.4 Full static state estimation

The full state estimation was then implemented. The results are shown in Figures 3.20, 3.21, 3.22, 3.23 similarly to the intermediate state estimation. In particular, it seems that an error was introduced, as the results shown in Figure 3.21 differ from those obtained by the adaptive odometry. Indeed, the signals can be seen to be of slightly different shapes and shifted along time. The cause of this error was not solved at the time of the report. As such, the cost for the slack variable which would allow the velocity to influence the other states was set to zero in order to avoid having the other states being influenced. A key result is that the positions of the CoP for each foot illustrated in Figures 3.23 and 3.24 were found to indeed correlate with the results obtained by the adaptive odometry. The constraints allowing to compute these positions were directly given as the sum of forces and moments, meaning it is indeed possible to setup an optimisation problem in such a manner.

Figure 3.20: Comparison of estimated base position between the adaptive odometry and the full static state estimation
Figure 3.21: Comparison of estimated base velocity between the adaptive odometry and the full static state estimation

Figure 3.22: Comparison of estimated base angular velocity between the adaptive odometry and the full static state estimation

Figure 3.23: Comparison of estimated right foot CoP position between the adaptive odometry and the full static state estimation
3.3 Dynamic contact scenario

By the time of this report, the dynamic contact scenario was not completed yet. As such, it is unknown if the algorithm proposed in (2.54) is correct. The expected results were that the state estimation would still be likely to drift, but in a less pronounced way than when directly integrating the values of the IMU sensor. No other similar work was found to have been done previously, meaning it is difficult to determine if this would indeed be the case.
4 Discussion

From the results, a new implementation of the state estimation of the robot when considering static contact was achieved. This was completed using constrained optimisation. Aside the error appearing in Figure 3.21 of the estimation of the base velocity, which is believed to be caused by a local implementation issue as the base velocity was estimated correctly in the intermediate state estimation, the state estimation yielded expected results. As such, it is concluded that the full state estimation algorithm came to show that the optimisation could be written such that the physical and geometrical relations of the robot could be directly written as the constraints of the optimisation algorithm. The optimisation algorithm could then solve the problem by converging recursively to an estimate. The full state estimation algorithm also has the advantage of estimating all CoP variables, i.e. its position, its force and its moment.

A properly working dynamic state estimation was not yet achieved during this project due to time constraints. It is still expected that this algorithm could solve and yield an appropriate estimation of the state if implemented properly. However, it is not expected that the state estimation would be fully tolerant to drift. Still, it would be expected to be less prominent than when considering IMU integration alone. In reality, this drift is mostly caused by the presence of noise in the measurements when using sensors that cannot locate the robot in the global frame. The key element of the dynamic state estimation proposed was that it is expected to also work when one foot is not directly in contact with the ground as the general dynamical relations were considered. Indeed, if the foot were to be above the ground, typically when performing a step, the CoP force and moment would be expected to be estimated to zero allowing the foot to move in any direction while still satisfying the constraints from Equation (2.40). The drift could then be expected to be limited by the foot still in contact with the ground. Note that if both feet were off the ground, then the state estimation would be equivalent to obtaining the global movement of the robot using the IMU only. In addition, the designed algorithm only works as long as the contact with the foot is only found to be in the bottom surface contact surface, as the CoP position was expected to be found there.

In the case where drift would need to be fully avoided, drift-free sensors could be added to the robot. This kind of approach was typically proposed and implemented by [29], where they used a LIDAR sensor to get rid of drift in the state estimation. Furthermore, the implementation of state estimation when using a constrained optimisation approach makes it convenient to easily add an extra drift-free sensor as another constraint. Additionally, this approach is quite interesting as drift-free sensors usually offer smaller refresh rates than IMUs, joint encoders or force/torque sensors. Indeed, in the case where different refresh rates are used, the constrained optimisation could still be implemented by simply setting the corresponding error costs inside the objective function to zero for the sensors that have not been refreshed, allowing to make use of the faster sensors over small periods of time to continue estimating the state. The estimation would then be corrected once a new measurement of a drift-free sensor is obtained.

Note that the use of constrained optimisation for state estimation is also applicable to other types of robots, including 4 legged robots or flying drones. This framework is quite interesting as it is even expected to be a convenient approach for most dynamical system in general, where the system would simply need to be modelled using the physics equations, geometrical constraints and other assumptions that can be made about the system. The goal is then to compute the state of the system by minimising the errors made from the measurements and predictions.

Finally, in the case where the dynamic algorithm were to be implemented successfully and if drift were to still be present, a proposition would be to add an IMU on each foot. Indeed, as previously explained in Section 2.5.3, the dynamics of the foot are computed from the base IMU, meaning that the measurements of the joints also need to yield the estimated velocities and accelerations of the foot in the base frame. Due to the resolution of the encoders, this may be expected to be a cause of inaccurate results. An IMU at the position of the foot would be capable of bypassing these joints, which may help obtain better results.
5 Conclusion

In this project, the implementation for the state estimation of a coman robot was studied. Furthermore, the complexity of the project required to design specific tools. In particular, a GUI environment for the visualisation and control of the robot data was implemented, as well as an automatic code generation script to generate code directly from the equations set.

The general algorithm proposed to obtain this estimation was non-linear constrained optimisation. This approach is similar to the Kalman filter if the objective function is set to minimise the errors of the measurements and predictions weighted with their respective error’s covariance matrix. It also has the advantage of dealing with non-linear hard and soft constraints. This makes this algorithm quite interesting, as it makes it possible to design the state estimation by directly giving the physical and geometrical relations.

During this project, the coman robot was not available for use. The state estimation was then implemented in the Webots simulator. The provided framework contained a model of the robot which would simulate the robot’s dynamics and sensors. Moreover, it was expected that the implementation conducted in the simulator could directly be sent onto the real robot by connecting it to Webots.

The state estimation was implemented by considering two scenarios. The first was that the contacts of the feet with the ground were static, meaning no slippage would occur. This allowed to compute the position of the robot by using this contact as a reference to the global frame. Thus all the positions and orientations of the different parts of the robot, as well as their respective velocities could be computed using the joint angles. The addition of using the IMU in this case could also lead to increased accuracy when considering the joint encoders to have poor resolution. The second scenario was one where slippage was considered. As such, the idea was that the work caused by friction with the ground at the CoP may only dissipate energy, and that the velocity at this position of the robot would need to be collinear to the estimated force.

The approach used for the project was to reimplement a part of the state estimation algorithm proposed by [1] using different optimisation libraries, and then to iterate the development of the state estimation from the static scenario to the dynamical one. As such, multiple constrained optimisation problems were proposed in order to yield the state estimation based off sensory data. The results obtained showed that the use of alternative libraries was successful at yielding similar results. On the other hand, the dynamic state estimation was not completed due to lack of time. As such, the dynamic contact algorithm remains to be tested in order to evaluate its potential in being used for state estimation.

Additionally, the initial simulation model did not implement any noise model for the sensors. As such, noise models were added manually. This showed to propagate noise in to the estimation of the state. This means that the objective function costs of the optimisation algorithms must be dimensioned correctly in order to yield an optimal estimation. As the algorithms proposed showed similar behaviour to the implementation conducted by [1] which was tested successfully on the coman robot, it was expected that the functioning algorithms might still function as long as the costs of the optimisation problems would be set up properly.

Finally, it is hoped that this project has laid a good enough foundation for any similar future projects to benefit from the methods and suggestions. Constrained optimisation was used in this project for state estimation, but it may also be used for identification and prediction of systems. It has the capacity to directly treating the hypothesis made for a system by writing them as as constraints. Moreover, its recursivity makes it suitable online computations, especially in the case of dynamical systems where the result of a new iteration is likely to be similar to the previous one. As such, constrained optimisation is a very useful tool in the field of robotics.
References


A Theory

A.1 Coordinate transformations

A.1.1 General definitions

Cross product:

$$\omega \times \leftrightarrow S(\omega) \rightarrow \text{where } \times \text{ is the cross-product and } S(\cdot) \text{ is the skew symmetric matrix}$$  \hspace{1cm} (A.1)

$$S(\omega)\omega = \omega \times \omega = 0$$  \hspace{1cm} (A.2)

$$a \times b = -b \times a$$  \hspace{1cm} (A.3)

Rotation matrix from frame a to b:

$$R^b_a$$  \hspace{1cm} (A.4)

Rotation matrix:

$$RS(\omega)R^T = S(R\omega)$$  \hspace{1cm} (A.5)

A.1.2 Rotation derivations

From frame a to b:

$$\dot{R}^b_a = S(\omega^b_a)R^b_a = R^b_aS(\omega^b_a)$$

$$\Rightarrow S(\omega^b_a) = R^b_aS(\omega^b_a)R^b_0 = S(R^b_0\omega^b_a)$$

$$\Rightarrow \omega^b_a = R^b_0\omega^b_a$$  \hspace{1cm} (A.6)

$$\ddot{R}^b_a = S(\dot{\omega}^b_a)R^b_a + S(\omega^b_a)S(\omega^b_a)R^b_0$$

$$= R^b_0S(\omega^b_a)S(\omega^b_a) + R^b_0S(\omega^b_a)$$

$$\Rightarrow S(\dot{\omega}^b_a) + S^2(\omega^b_a) = R^b_0S(\omega^b_a)R^b_0R^b_0S(\omega^b_a)R^b_0$$

$$\Rightarrow \dot{\omega}^b_a = \alpha^b_a = R^b_0\alpha^b_a$$  \hspace{1cm} (A.7)

From frame a to c passing by b:

$$R^c_a = R^c_bR^b_a$$  \hspace{1cm} (A.8)

$$\dot{R}^c_a = S(\omega^c_a)R^c_a = S(\omega^c_b)R^c_bR^b_c + R^c_0S(\omega^b_a)R^b_0$$

$$\Rightarrow S(\omega^c_a) = S(\omega^c_b) + R^c_0S(\omega^b_a)R^b_0$$  \hspace{1cm} (A.9)

$$\Rightarrow \omega^c_a = \omega^c_b + R^c_0\omega^b_a$$

$$\alpha^c_a = \alpha^c_b + S(\alpha^c_b)R^c_b\omega^b_a + R^c_0\alpha^b_a$$  \hspace{1cm} (A.10)
From frame a to c passing by b in local frames:

\[
\dot{R}^c_a = R^c_a S(\omega_{a}^{c,a}) = R^c_b S(\omega_{b}^{c,b})R^b_c + R^c_a R^b_c S(\omega_{a}^{b})
\]

\(\Rightarrow S(\omega_{a}^{c,a}) = S(R^c_b \omega_{b}^{c,b}) + S(\omega_{a}^{b,a})\)  

\(\Rightarrow \omega_{a}^{c,a} = R^c_b \omega_{b}^{c,b} + \omega_{a}^{b,a}\)  

\(\alpha_{a}^{c,a} = R^a_b S(\omega_{b}^{a,b})\omega_{b}^{c,b} + R^a_b \alpha_{b}^{c,b} + \alpha_{a}^{b,a}\)  

(A.12)

### A.1.3 Position derivations

**Position:**

\[P_a^c = P_b^c + R_b^c P_a^b\]  

(A.13)

**Velocity:**

\[v_a^c = v_b^c + R_b^c v_a^b + S(\omega_{a}^b v_a^b)\]

\[= v_b^c + R_b^c v_a^b + S(\omega_{a}^b (P_b^c - P_a^c))\]

(A.14)

**Acceleration:**

\[a_a^c = a_b^c + 2S(\omega_{a}^b v_a^b)R_b^c v_a^b + S(\omega_{a}^b)R_b^c P_a^b + S(\omega_{a}^b)R_b^c P_a^b\]

\[= a_b^c + R_b^c a_a^b + S(\omega_{a}^b)R_b^c v_a^b + S(\omega_{a}^b)R_b^c v_a^b + S(\omega_{a}^b)(v_a^c - v_b^c)\]

(A.15)

### A.1.4 Jacobians

With q representing the joint angles:

\[P_e^i = P_b^i + R_b^i P_e^{i}(q)\]  

(A.16)

\[v_e^i = v_b^i + R_b^i v_e^b + S(\omega_{b}^i v_e^b)\]

\[= v_b^i + R_b^i \frac{\partial P_e^{i}}{\partial q} \dot{q} + S(\omega_{b}^i)R_b^i P_e^{i}\]

\[= S(R_b^i \omega_{b}^i)R_b^i P_e^{i}\]

\[= R_b^i S(\omega_{b}^i)R_b^i P_e^{i}\]

\[= S(\omega_{b}^i)P_e^{i}\]

\[= -S(P_e^{i})\omega_{b}^i\]

(A.17)

\[V_e^i = \left[ I - S(P_e^{i}) \frac{\partial P_e^{i}}{\partial q} \right] \begin{bmatrix} v_b^i \\ \omega_{b}^i \\ \dot{q} \end{bmatrix}\]  

(A.18)

\[\omega_e^i = \omega_b^i + R_b^i \omega_e^b\]

\[= \omega_b^i + R_b^i \frac{\partial P_e^{i}}{\partial q} \dot{q}\]

\[= R_b^i \omega_b^i + R_b^i \frac{\partial P_e^{i}}{\partial q} \dot{q}\]

(A.19)
\[
\v_i^e = R_i^e \left[ R_i^e - R_i^e S(P_b^e) \right] R_b^e \frac{\partial R_b^e}{\dot{q}} \left[ \begin{array}{c} v_b^i \\ \omega_b^i \end{array} \right]
\]
(A.20)

\[
\omega_b^i = R_i^e \left[ \begin{array}{cc} 0 & R_i^e \frac{\partial R_b^e}{\dot{q}} \\ R_i^e & \omega_b^i \end{array} \right] \left[ \begin{array}{c} v_b^i \\ \dot{q} \end{array} \right]
\]
(A.21)

**A.1.5 Jacobian decomposition**

**Position:**

\[
P_i^f = P_i^b + R_i^b P_f^b
\]
(A.22)

**Velocity:**

\[
v_f^i = v_b^i + R_i^b S(\omega_b^i) P_f^b + R_i^b v_f^b
\]
\[= v_b^i - R_i^b S(P_f^b) \omega_b^i + R_i^b \frac{\partial v_b^i}{\dot{q}} + R_i^b \frac{\partial \omega_b^i}{\dot{q}} \]
\[= \begin{bmatrix} I & -R_i^b S(P_f^b) & R_i^b \frac{\partial \omega_b^i}{\dot{q}} \end{bmatrix} \begin{bmatrix} v_b^i \\ \omega_b^i \\ \dot{q} \end{bmatrix}
\]
(A.23)

**Orientation:**

\[
R_i^f = R_i^b R_f^b
\]
(A.24)

**Angular velocity:**

\[
S(\omega_f^i)R_i^f = S(\omega_b^i)R_i^b R_f^b + R_i^b S(\omega_f^b) R_f^b
\]
\[= S(R_i^b \omega_b^i) R_i^b R_f^b + S(\omega_f^b) R_i^b R_f^b
\]
\[\Rightarrow \omega_f^i = R_i^b \omega_b^i + R_i^b \omega_f^b
\]
(A.25)

\[
\omega_f^i = R_i^f \left[ \begin{array}{cc} 0 & R_i^f \frac{\partial R_f^b}{\dot{q}} \\ R_i^f & \omega_f^i \end{array} \right] \left[ \begin{array}{c} v_f^i \\ \dot{q} \end{array} \right]
\]
(A.26)
B Software installation

The project was completed using multiple software. First, Webots 7.2.1 was used as the simulator and was installed from the source files obtained from the Cyberbotics archive repository. Note that printing the output results to the Webots console was observed to slow down the simulation. A suggestion would be to launch webots by printing the output messages to an external console using `./webots --stdout`.

The model of the Coman robot was then provided by the Biorob lab using the following:

```
git clone -b devel git://cnbisrv02.epfl.ch:443/rtfilter.git
git clone -b devel https://github.com/nbourdau/rtfilter.git
sudo apt-get install libeigen3-dev
sudo apt-get install librtfilt3r-dev
sudo apt-get install libatlas-dev
sudo apt-get install monodevelop
sudo apt-get install libblas-dev
sudo apt-get install libatlas-base-dev
sudo apt-get install mesa-utils
sudo apt-get install freeglut3-dev
sudo apt-get install cmake
sudo apt-get install libboost-thread-dev
sudo apt-get install libboost1.48-dev
wget http://msgpack.org/releases/cpp/msgpack-0.5.4.tar.gz
tar zxfv msgpack-0.5.4.tar.gz
cd msgpack-0.5.4
./configure
make
sudo make install
```

The libcoman package would then have to be installed from the wbo/libcoman folder using the following commands:

```
./configure
make
sudo make install
```

Python 3.4 was embedded into the C++ controller in order to visualise results and interact with the robot. Multiple Python libraries were used in order to implement this. The Python packages could be obtained using the Python Package Index with the `pip` command. The versions of the packages were the following:

- Numpy 1.9.2
- Scipy 0.16.0
- Sympy 0.7.6.1
- PySide 1.2.2-1
- PyQtGraph 0.9.10
- Matplotlib 1.4.3
### B.1 Makefile

The makefile used to compile the library was the following:

```makefile
include ../..../mf/webots-common.mf

PACKAGES = coman-webots-1.0
ALG_PATH = ../../algorithms
SCENARIO = s_arreguit

ifeq ($(shell pkg-config --exists $(PACKAGES)) && echo -n '1'),
$(error Not all required packages [$(PACKAGES)] are installed. Please
  → install all the dependencies and try again.)
endif

CFLAGS=-I$(LIBCOMAN_DIR) ../ -std=c++0x -I$(WEBOTS_HOME)/include/controller
  → /c -I$(ALG_PATH)/robot/include -I$(ALG_PATH)/$(SCENARIO)/include $(
  → shell pkg-config --cflags $(PACKAGES))
LIBRARIES=+lController +lboost_system +lrobot +l_odo +l_sdfast -l_WBO -L$(
  → ALG_PATH)/robot/build/lib/ -L$(ALG_PATH)/$(SCENARIO)/build/lib/$(
  → shell pkg-config --libs $(PACKAGES))

# Add SNOPT7 solver library
CFLAGS += -I$(SNOPT)/interfaces/
LIBRARIES += -Wl,-rpath $(SNOPT)/lib/ -L$(SNOPT)/lib/ -lsonopt7_cpp -lsonopt7

# Add Python support
CFLAGS += -I/usr/include/python3.4m
LIBRARIES += -lpython3.4m

# CC_SOURCES = $(wildcard *.cc)
CNX_SOURCES = $(wildcard *.cc) $(wildcard $(ALG_PATH)/$(SCENARIO)/src/*cpp
  → ) $(wildcard $(ALG_PATH)/$(SCENARIO)/src/*cpp)
C_SOURCES = $(wildcard $(ALG_PATH)/$(SCENARIO)/src/*c) $(wildcard $(
  → ALG_PATH)/$(SCENARIO)/src/*c)

include ../..../mf/webots-controller.mf
```

In particular, the following lines from the makefile correspond to the flags and libraries necessary in order to run the wbo project with SNOPT and Python.

```makefile
# Add SNOPT7 solver library
CFLAGS += -I$(SNOPT)/interfaces/
LIBRARIES += -Wl,-rpath $(SNOPT)/lib/ -L$(SNOPT)/lib/ -lsonopt7_cpp -lsonopt7

# Add Python support
CFLAGS += -I/usr/include/python3.4m
LIBRARIES += -lpython3.4m
```

The flags necessary for embedding Python 3.4 may vary depending on the installation. The appropriate CFLAGS can typically be obtained using `pythonX.Y-config --cflags`. The SNOPT path must also be added to the bash path appropriately.
C Code

C.1 Brief files descriptions

For the project, a new scenario folder was created and named _s_arreguit_. All the files generated during the development of the project can be found in this folder. The main files include are the following:

- **include**
  - Scenario.hpp: Header for both the scenario.cpp file and the state_estimation.cpp file.
  - py_control.hpp: Header for the py_control.cpp file.
  - state_solver.hpp: Header for the state_solver.hpp file.
- **src**
  - Python
    - * Problems: Contains the SNOPT code generators for the different state estimation algorithms
    - SE_GUI: Contains files necessary for the GUI using PySide.
    - SE_IO: Contains the management of the input/outputs for the exchange of data between the C++ code and the Python code.
    - SE_Optimisation: Contains the reimplementation of the adaptive odometry algorithm in Python using Scipy.
    - SE_Robot: Contains the Robot class to structure the data obtained from the C++ code.
    - Saved_Plots: Folder containing the plots generated from the Python GUI.
    - sepycore.py: Main file for the interaction between the C++ and the Python code. The C++ typically calls the py_interface function in order to read and write data. The Python GUI process is also spawned from this file. A inter-process pipe is used in order to send the data object from the Robot class to the GUI.
  - Scenario.cpp: Contains the typical Scenario
  - py_control.cpp: Contains the function calls for the exchange of data between the C++ code and the Python code.
  - state_estimation.cpp: File containing the call for all of the state estimation algorithms.
  - state_solver_dynamic.cpp: Implementation of the dynamic state estimation using SNOPT.
  - state_solver_static.cpp: Reimplementation of the adaptive odometry algorithm using SNOPT.
  - state_solver_static_full.cpp: Full static contact state estimation algorithm using SNOPT.
  - state_solver_static_inter.cpp: Intermediate static contact state estimation algorithm using SNOPT.
C.2 Code generation

The code generators were written as python scripts. They would output the code in a console, which could then be copy-pasted into the C++ file for the SNOPT optimisation call. Examples can be found in the code. In particular, some small modifications must be done manually in order to launch the code. The first is that the square powers must be rewritten from #1**2 to #1*#1 for C++, the second modification is that the eigen library uses parenthesis instead of brackets to designate matrix indices. As such, they must also be replaced. This can be done quite fast when using search and replace methods with regular expressions, which could be used as follows in the KATE editor. Other text editors are also expected to hold equivalent features.

<table>
<thead>
<tr>
<th>Find:</th>
<th>Replace:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(.....) **2</td>
<td>\1*\1</td>
</tr>
<tr>
<td>[(., .)]</td>
<td>(\1, \2)</td>
</tr>
</tbody>
</table>

Example: x[0]**2 becomes x[0]*x[0] and [0, 1] becomes (0,1)