Technical Documentation for Bolt Estimator

Niels Albrecht

March-August. 2024

Abstract

This document aims at providing explanations, documentation & technical details on the Bolt bipedal robot center of mass estimator, as conceived in 2024. If you need more details on what this is all about, refer to section 10 and 11.

Contents

1 Introduction and goals

1.1 Goals

Bolt is a legged robot [1] [2]. As such, it is mobile and is supposed to be walking around on different types of terrain. During its time moving, Bolt has to maintain balance and some notion of where it is and where it is going. If it fails to do so, it will not be able to compute where its feet should be to *a)* keep upright and *b)* go where one wants it to go.
For simplicity's sake, the base of the robot speed, acceleration, rotation etc will be denoted as the robot's speed, acceleration, rotation etc.
Bolt do include sensors to give it some sense of where it is. These are, namely :

- a 9-axis IMU [3]
  Includes an accelerometer, a gyrometer, a magnetometer, and a built-in Kalman filter.
- encoders
  On every joints except ankles (6 in total).
- current sensor
  On every motor (6 in total)

Those sensors give us :

- Instantaneous acceleration and rotation speed of Bolt's base
  $\hookrightarrow$ acceleration & rotation speed
- 6 joints angle
  $\hookrightarrow$ position and attitude of base in foot frame
- current in each motor
  $\hookrightarrow$ torque

Based on this, we should be able to deduce Bolt's location, speed and acceleration. However, it is not straightforward, because all of the aforementionned datas are noisy and wrong to some extent. Plus, the position is only that of the feet of the robot to its base, and we need its position with regard to the world frame. Worse, the feet are rounded, and we have no direct measurement of the angle the foot makes with regards to the ground surface.
The aim of the estimator is to derive the speed, rotation speed, and acceleration of Bolt based on these noisy, imprecise, and possibly wrong datas.

1.2 Notations

Here are how all parameters relative to the estimator are denoted in this document. These notations should be the same as those used in the code of Bolt estimator. When different, the code version is between parenthesis.

- c
- $\dot{c}$ (or cdot, cd)
- $\ddot{c}$ (or cdotdot, cdd)
- a
- ω (or w or $\dot{\theta}$)
- θ (or theta)
- z (resp x, y)
- q is the base's position concatenated with the base's orientation (as quaternion) and the joints angles. $q = [p, q, \theta_1 ... \theta_6]$. It is in $\in \mathbb{R}^{13}$
- $\dot{q}$ (or qdot) is the base's speed concatenated with the base's angular speed ($\in \mathbb{R}^3$) and the joints angular speed. $q = [p, q, \theta_1 ... \theta_6]$. It is in $\in \mathbb{R}^{12}$
- $\ddot{q}$

- R
- τ (or tau)

If data x is extracted from IMU data, it will be denoted x_{imu} (or x_imu).
If data x is extracted from forward kinematics, it will be denoted x_{fk} (or x_fk).
If data x is from the Tilt subestimator, it will be denoted x_{tilt} (or x_tilt).
If data x is the data to be used externally (for instance, the output of the estimating and filtering stages), it will be denoted x_{out} (or x_out).

1.3 Specifications

The estimator should take into account IMU data, encoders data, and current data. It should give an accurate estimation of :

- Bolt's speed
- Bolt's attitude
- Bolt's rotation speed
- Bolt's position
- Which leg(s) of Bolt are in a reliable contact with the ground
- The contact forces on each feet

The estimator has to be modular and adaptable to other robots with minimal work. It must be in the form of a set of functions and objets, easy to adapt or rewrite.
It is to be tested in Python, then in C++, to be in the end packaged in ROS2 Control.

1.4 Specifications reached and failed

1.5 Followed path

2 Architecture of the proposed Estimator

2.1 Overall Architecture

The schematics below summarize the different objects and the exchange of data between them.

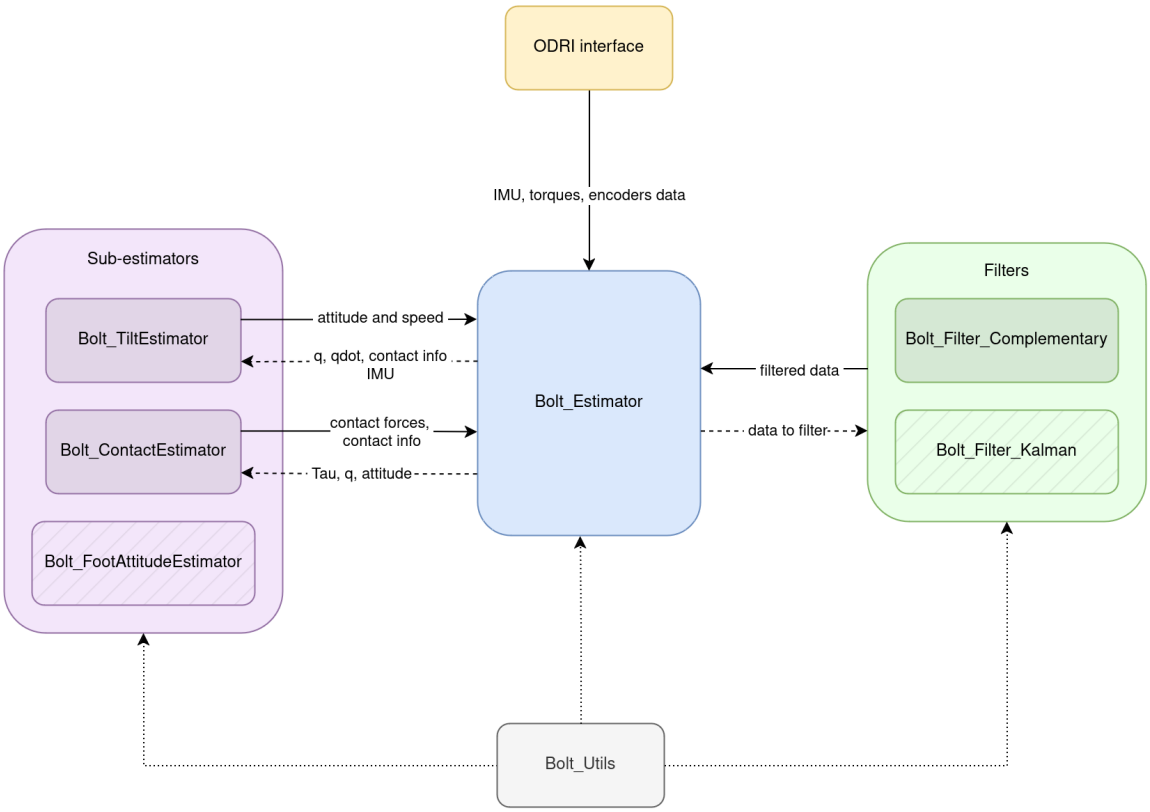


Figure 1: The structure of Bolt estimator

Below is a list of code objects used and an overview of how they interact with one another.

- *Bolt_Estimator*
  Reads the data from sensors, provided by an interface. Initializes and runs filters. Runs *ContactEstimator* and *TiltEstimator*. Merges and filters the obtained data to provide a, ω, c, ...

- *Bolt_ContactEstimator*
  Reads data provided by *Bolt_Estimator*. Derives contact forces on each foot.

- *Bolt_TiltEstimator*
  Reads data provided by *Bolt_Estimator*. Estimate base attitude and speed in world frame, robot-oriented.

- *Filter_Complementary* and *Filter_Kalman*
  Filters data provided by estimators or test programs, and returns it.

- *Bolt_Utils*
  Provides a set of objects and functions, such as a logger, to be used in several other objects.

- *TrajectoryGenerator* and *Metal*
  Generates a trajectory following user input, with the time derivatives that come with it. It can also read an existing trajectory and derives its time derivatives. It provides a true set and a noisy set of trajectory, speed, acceleration. *Metal* adds noise to a provided trajectory.

- *Graphics*
  Takes a set of 1D, 2D or 3D trajectories and plots it, with semi-automatic graph settings.

2.2 Estimator Detailed Architecture

Below is the Estimator's structure. Compared to previous figure, this one focuses on the Estimator, Contact Estimator and Filter objects. It details their methods. The filters can be of diffent types, such as Bolt_Filter_Complementary CITE or Bolt_Filter_Kalman.

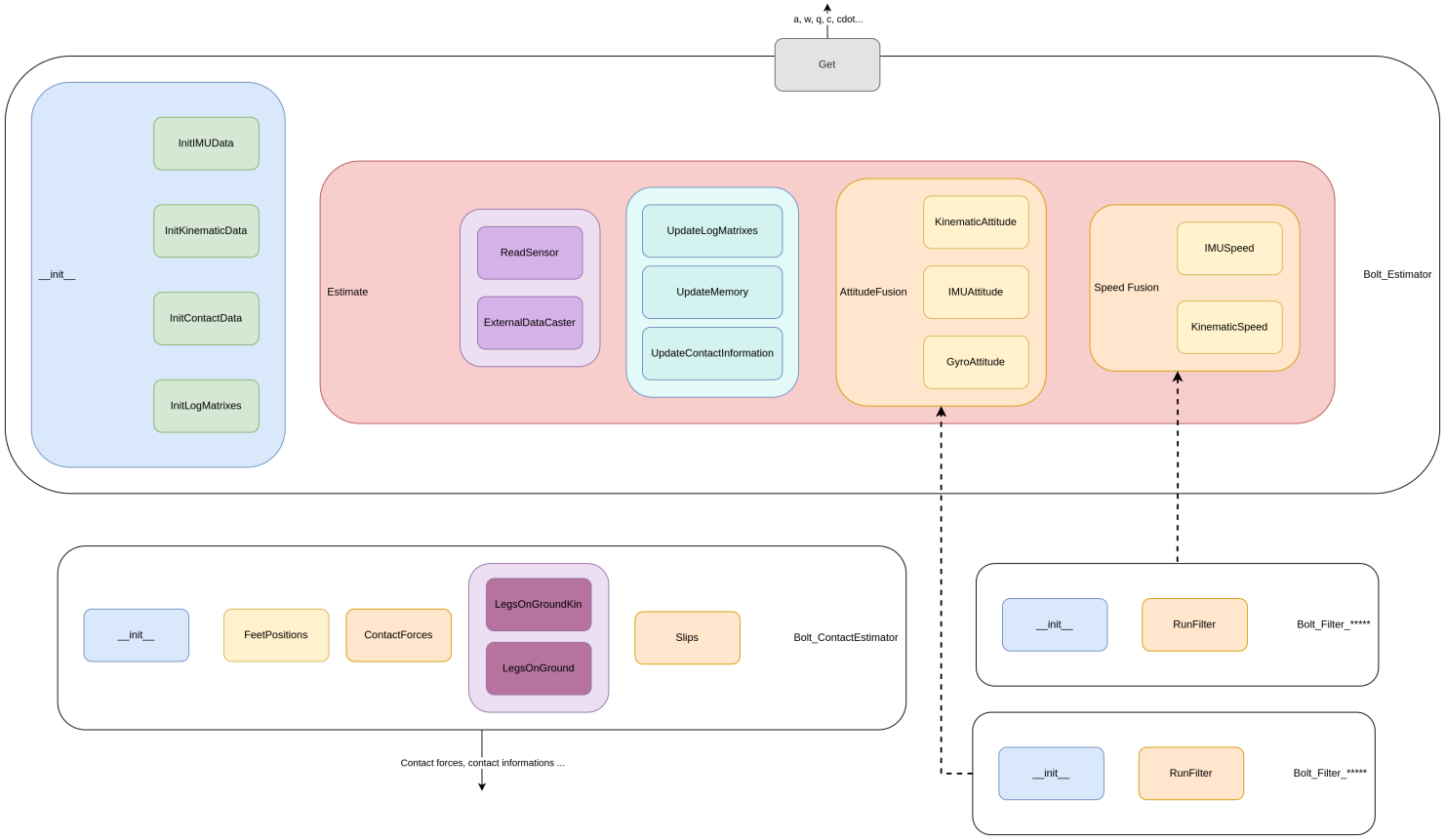


Figure 2: Estimator architecture

3 Filtering

The proposed estimator needs three distincts filters. These filters might, or might not, have the same specifications and parameters. They are :

- The Attitude filter
  uses θ_{tilt} as quaternion and ω_{imu}
  $\hookrightarrow$ gives us $\theta_{filtered}$
- The Speed filter
  uses a_imu and v_tilt
  $\hookrightarrow$ gives us v_{out}
- The Base Height filter
  uses v and c

$\hookrightarrow$ gives us c_z

It is of course possible to use one multi-dimensionnal filter that will do the job of both filters in one go. However, that implies we choose one type of filter for the two different filtering operations. Using 3 different filters proved to be more modular and practical for tuning.

3.1 Filter options comparison

The best filter we can get for solving this type of problem would be a Kalman filter. It is supposed to be fast and very stable of all. However, Complementary filters are much simpler to tune and provides already accurate results. Solo state-of-the-art estimator is based on a complementary filter. It has to be noted that the 3DM-CX5-AHRS IMU of Bolt and Solo have a built-in Kalman filter, so any filter in our estimator will be added on top of this one. Because it is much simpler to implement and tune, more flexible and has been recommended by both T.Fayols and O.Stasse, the first implemented filter will be a complementary one. Later on, the estimator will offer an array of options for filtering, including at least a Kalman filter and a Complementary filter. To keep this option, Estimator accept filter parameters as list so that the number of parameters can vary depending on the type of used filter.

3.2 Complementary Filter principle

In this section :

- a is a frequency
- T is a time constant
- x is the true value of what we want to filter
- x^* is the noised-out x that serves as input
- $\widetilde{x}$ is our filter output and estimate for x
- c_{og} is the integrator / offset gain

A complementary filter (CF) is based on the following scheme [4]:

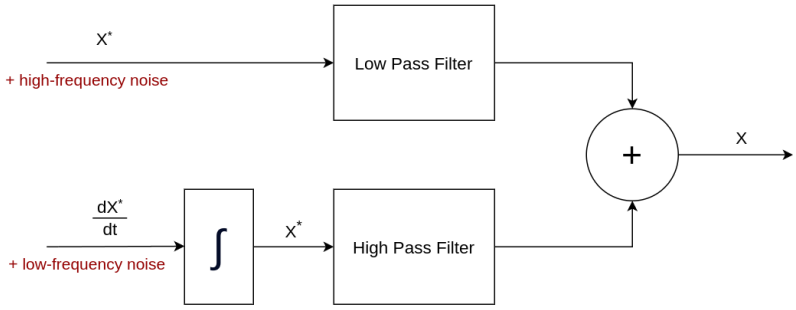


Figure 3: Complementary Filter

We can use first-order low-pass and high-pass filters, whose equations are

$$LP(s) = \frac{1}{1+as}$$

$$HP(s) = \frac{as}{1+as}$$

then

$$\widetilde{x} = \frac{1}{1+as} \cdot x^* + \frac{as}{1+as} \cdot \frac{1}{s} \cdot \dot{x}^*$$

thus

$$(1 + as) \cdot \tilde{x} = x^* + a \cdot \dot{x}^*$$

with Laplace inverted transform, we obtain

$$\widetilde{x} + a \cdot \tilde{\dot{x}} = x^* + a \cdot \dot{x}^*$$

no longer omitting the time dependance of our functions, we can then approximate $\tilde{\dot{x}}$ with data we have :

$$\widetilde{x}(t) + a \cdot \frac{\widetilde{x}(t) - \widetilde{x}(t-T)}{T} = x^*(t) + a \cdot \dot{x}^*(t)$$

thus

$$\widetilde{x}(t) = \frac{a}{T+a} \cdot \widetilde{x}(t-T) + \frac{T}{T+a} \cdot x^*(t) + \frac{Ta}{T+a} \cdot \dot{x}^*(t)$$

denoting $b = \frac{a}{T+a}$ we get

$$\widetilde{x}(t) = b \cdot \widetilde{x}(t-T) + (1-b) \cdot x^*(t) + T \cdot b \cdot \dot{x}^*(t)$$

we have the output depending only on inputs and previous outputs.

The parameter a depends on the frequencies of the noise we want to cut, and is often chosen at a few Hz. T depends on the refresh rate of the filter. After implementing this filter along with a trajectory generator, here is what we get :

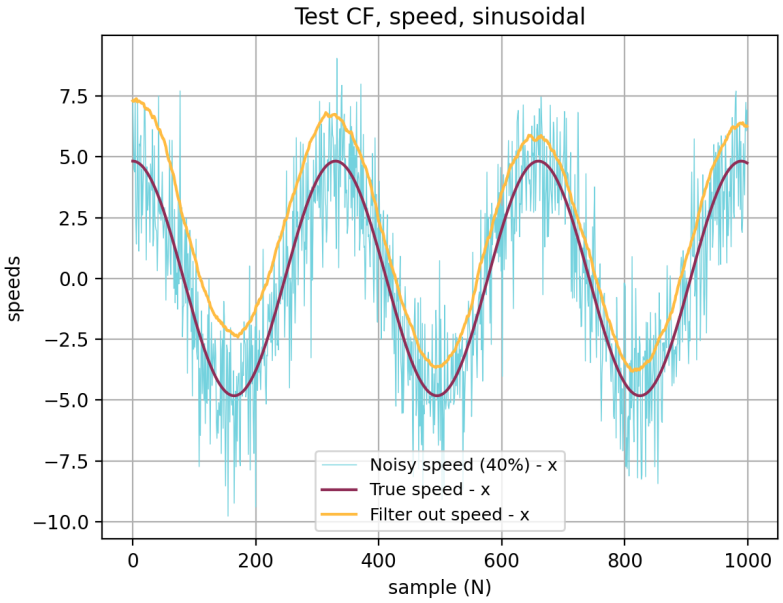


Figure 4: Simple complementary filter run

The generator generate the purple sinus curve and its derivative, corresponding to speed and acceleration. The signals are composed of 1000 points. It then add a white gaussian noise to both. The noisy speed and acceleration is the input of the CF.

3.3 Complementary Filter with integrator

It appears that the filter output (yellow) is close enough in shape to the purple targeted signal. However, there is an offset between the signals. To address this issue, an option is to average the noisy signal and the filter output over ΔT, and consider that the difference between those averages is the offset we want to get rid of. ΔT has to be chosen long enough for the noise to cancel itself out, and small compared to a characteristic time of the signal. On the following exemples, this ΔT is arbitrarily set around 0.1s. This acts as a pseudo-integration of the error. In discrete-time, it changes the output as follows :

$$\widetilde{x}_k \mapsto \widetilde{x}_k + c_{og} \cdot M_k^N$$

where

$$M_k^N = \sum_{j=k-N}^{k} \widetilde{x}_j - \sum_{j=k-N}^{k} x_j^* = \sum_{j=k-N}^{k} (\widetilde{x}_j - x_j^*)$$

is the integrated error over the past N samples.

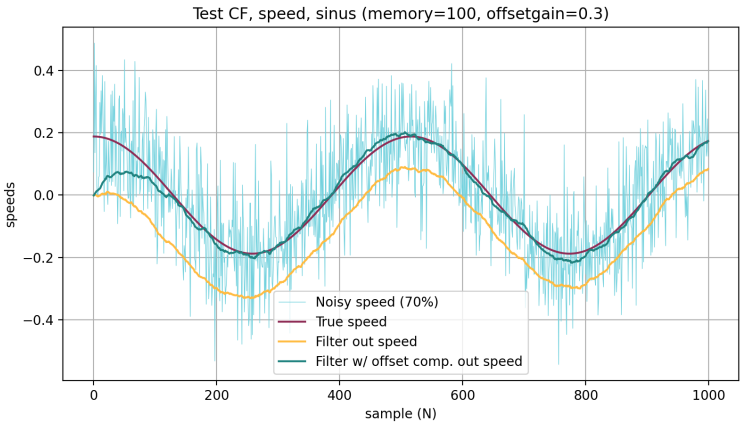


Figure 5: Complementary filter run, with and without offset

In most cases, this filter successfully get rid of the unwanted offset. Yet its performances depends a lot on the chosen ΔT. This is most visible on very noisy, slow dynamics polynomial signals. Here, ΔT ranges from 0.02s to 0.3s.

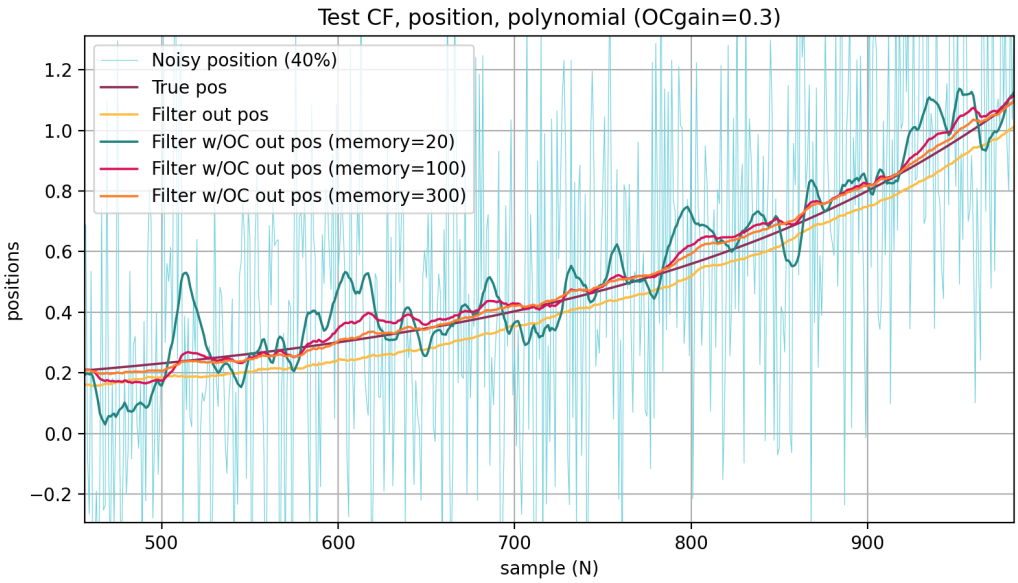


Figure 6: Complementary filter run, with different memory sizes

Still, on some signals, this pseudo-integration takes some time to overcome the bias completely. This time of integration depends on the memory size, but I could not reduce it below around 200 signal samples without having an noisy filter output.

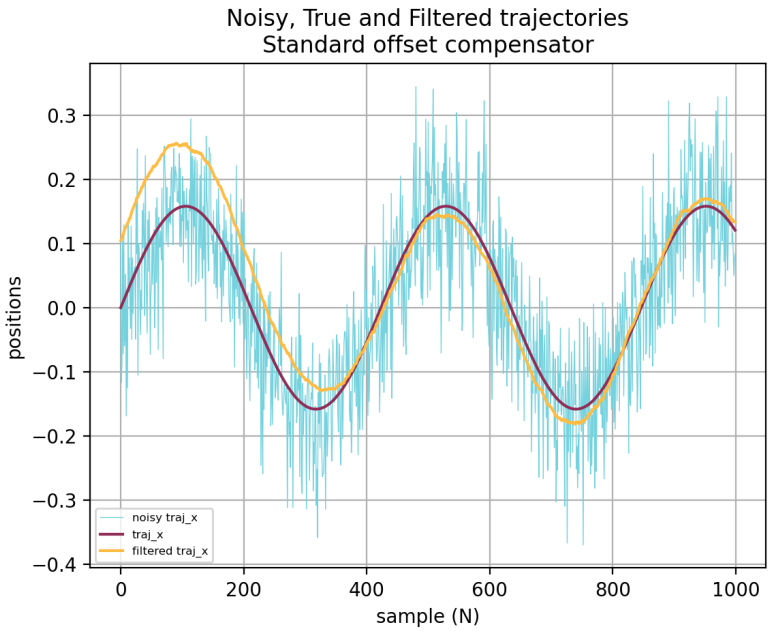


Figure 7: Error integration needs some time to take effect

To adress this issue, a time-varying gain was implemented. It works as follow.

$$\widetilde{x}_k \mapsto \widetilde{x}_k + c_k^{quickgain} \cdot M_k^N$$

where M_k^N remains the averaged error over the N past samples, and

$$c_k^{quickgain} = max(c_{og}, (\frac{N}{\alpha} - k)\beta)$$

$$\alpha, \beta \in \mathbb{R}$$

It enables quicker convergence on the $\frac{N}{\alpha} - \frac{c_{og}}{\beta}$ first iterations. However, it is sensitive to a non-null average noise on that first period.

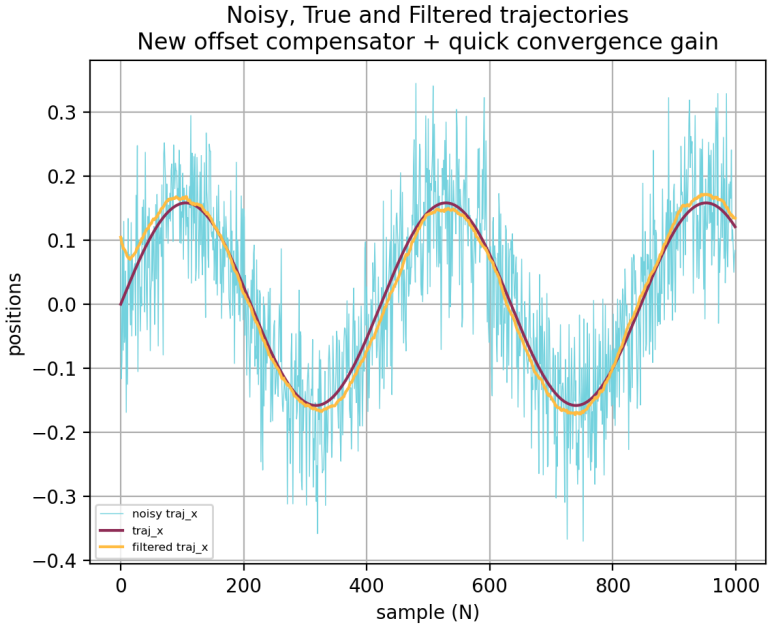


Figure 8: Same signal as previous figure, but with adaptative gain

Let's check that this filter with pseudo-integration is able to perform its function on signals with drifting derivatives.

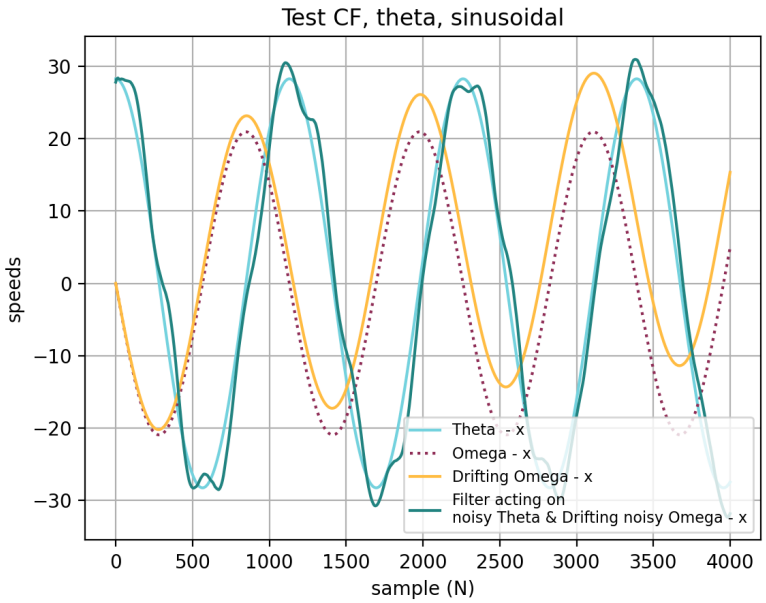


Figure 9: Filter with pseudo-integrator running on a signal with drifting derivatives

In this situation, we generate a signal (θ) and its derivative (ω). We then add a drift to ω : its is no longer a function like $sin(x)$ but rather $x + sin(x)$. On top of that, blank noise is added to both θ and ω before they are sent to the filter. This process serves as a simplistic model for IMU data filtering, in which ω from the accelerometer has a high-frequency noise, but no drift, and θ from the gyrometer has a drift we have to make up for.

3.4 Complementary Filter for quaternion

In order to obtain faster computation and prevent singularities, the estimator uses quaternions [5] to describe attitude. However, angular speed are represented by a vector of $\mathbb{R}^3$. From our code point of view,

$$\theta_q \in \mathbb{R}^4$$
$$\omega \in \mathbb{R}^3$$

However, our complementary filter needs x^* and $\dot{x}^*$ to be of same dimension. We need to adapt our filter to work properly in this situation.

We denote

$$\theta_q = \begin{bmatrix} q_x \\ q_y \\ q_z \\ q_\omega \end{bmatrix}$$
$$\omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

And we filter θ_q with its time derivative CITE

$$\dot{\theta}_q = \omega_q = 0.5 \cdot \begin{bmatrix} 0 & -\omega_x q_x - \omega_y q_y - \omega_z q_z \\ \omega_x q_\omega + & 0 \quad + \omega_z q_y - \omega_y q_z \\ \omega_y q_\omega - \omega_z q_x + & 0 \quad + \omega_x q_z \\ \omega_z q_\omega - \omega_y q_x - \omega_x q_y + & 0 \end{bmatrix} \in \mathbb{R}^4$$

with the exact same method described in 3.2, with $x^* = \theta_q$ and $\dot{x}^* = \omega_q$ of dimension 4.

3.5 Implementation of complementary filter

As 3.3 and 3.4 explained different modifications of the filtering process, these are implemented in the *Bolt_Filter_Complementary* class. Below are the method to run the filter from which to chose.

- Runfilter : run the filter
- RunFilterQuaternion : run the filter on a quaternion and a angular speed
- RunFilterOffset : run the filter with a pseudo-integrator
- RunFilterOffsetAdaptative : run the filter with a pseudo-integrator and a stronger gain in the first iterations

A *Bolt_Filter_Complementary* object keeps its last estimate as a state variables and store various logs. It necessary that a given filter is used always on the same data, for instance not on speed and then on attitude.

4 Estimating Tilt and speed

4.1 Goals

The IMU provides us the sum of earth's gravity and robot's base acceleration. From this sum, we want to obtain the direction of earth's gravity, and from this direction we can derive the attitude of the robot'base. Addtionnaly, the IMU includes a gyrometer which gives us access to the first-order time derivative of the attitude. This is suitable for a complementary filter implementation.
The objective is to reconstruct the attitude of the robot's base based on all this.

4.2 Implementation

The code used is an implementation of [6] The code file is named Bolt_TiltEstimator and is composed of functions:

- init
- InitLogs and UpdateLogs
  Can be disabled.
- SetInitValue function
  It enables one to change the start truth to prevent bias due to initial conditions.
- PinocchioUpdate
  This function centralize all calls to Pinocchio. It updates every data from Pinocchio that might be used during computation. For instance, it updates the attitude and speed of the base in contact foot frame.
- ErrorUpdate
  Update an estimate of the errors on state variables as they were defined in the paper. Not used elsewhere in the code.
- Estimate
  This is the main function. It calls PinocchioUpdate, compute the state variable derivative, and integrate them. It then returns the state variables.

Please note :

- This function needs to know the foot touching the ground. It relies on contact information provided by Estimator.
- Logging can be disabled for less memory use and faster processing.
- Notation are the same as those used in the paper.
  x_1 is the speed, x_2 is a unite vector corresponding to the gravity direction.
- Two possibilities are mentionned by the author (in a yet to come update of the paper) to compute y_v. Both are implemented, only the first one is used.

Below are schemes about the implementation of tilt and speed estimation.

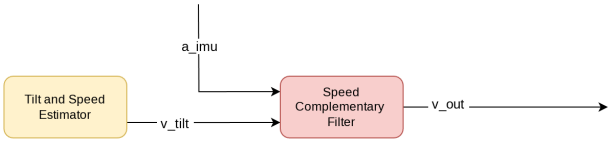


Figure 10: Speed datapipe

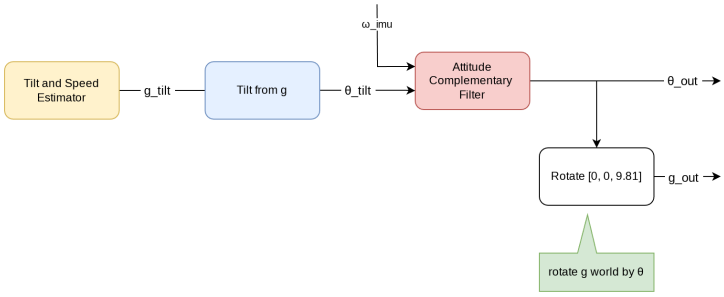


Figure 11: Tilt datapipe

5 Estimating Contacts

5.1 Goals

Bolt does not have force sensors in its feet. Therefore, Bolt doesn't know wether or not it is touching the ground. Moreover, it has no way of knowing the angle its rounded feet make with the ground. Knowing this can prove useful to correct or complete attitude measurements.

5.2 Structure

To estimate wether or not a foot is touching the ground, we proceed as follow.

1. Estimate the contact forces from legs torques, IMU acceleration, model dynamics. That gives us 3 different estimates.
2. Check that these forces are trustworthy (eg, coherent with one another).
3. Consider that the stronger the contact force, the most likely our contact is.
4. Return a boolean value based on the trust and the probability

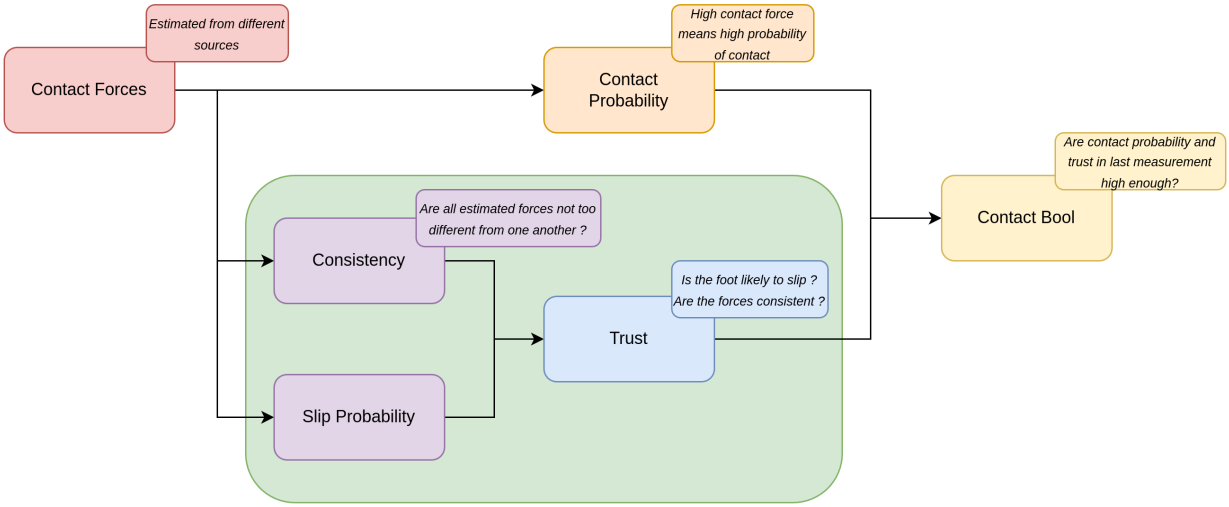


Figure 12: How ContactEstimator works

5.3 Contact Forces

We have different ways of estimating contact forces [7][8][9]. The implemented methods are as follow :

- ContactForce1D : Estimate the contact force on both feet.
  It considers that contact forces F_1, F_2 are vertical (which is untrue), each applied at the center of a foot (which should be close from true). Then, it considers that

$$F_1, F_2 = \underset{F_1,F_2}{\text{Argmin}}(\ddot{c}_{imu} - \ddot{c}_{fk}(F_1, F_2, \tau, \dot{q}))$$

  We obtain $F_i^{1D} = \begin{bmatrix} 0 \\ 0 \\ F_i^z \end{bmatrix}$

- ContactForce3D : Estimate the contact force on both feet.
  It uses the fact that if one neglect the legs inertia and speed [10] [8], one has

$$F_c = (J_c^T)^{-1} \cdot (\tau - b - g)$$

  b being the nonlinear term and g the generalized gravity.

- ContactProbability_Torque : Estimate the contact force and contact probability on both feet.
  This function is simple and takes advantage of the geometry of Bolt. We use the attitude estimate and the encoders to compute the horizontal knee-foot distance d. Then, we divide the knee torque by d. We thus obtain the contact force.

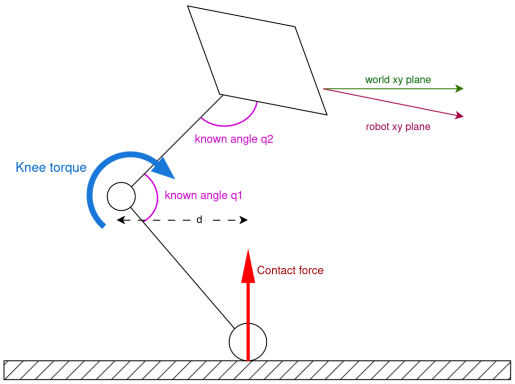


Figure 13: Contact probability using knee torque only

At this point, we have different estimates for our contact forces. We then introduce a sigmoid probability function as follows :

$$\begin{aligned} P_s : \mathbb{R}_+ &\to [0;1] \\ x &\mapsto \frac{1}{1+e^{-b\cdot x+b_0}} \qquad b, b_0 \in \mathbb{R}_+^* \end{aligned}$$

Which maps any real data to a $[0;1]$ interval of probability. And considering that

$$\begin{aligned} P_s(x_{center}) = 1/2 \quad &\Leftrightarrow \frac{1}{2} = \frac{1}{1+e^{-b\cdot x_{center}+b_0}} \\ &\Leftrightarrow 1 = e^{-b\cdot x_{center}+b_0} \\ &\Leftrightarrow b\cdot x_{center} = b_0 \end{aligned}$$

we will tune the parameters x_{center} and b_0 by hand, and compute b such that $P_s(x_{center}) = 1/2$, ie

$$b = \frac{b_0}{x_{center}}$$

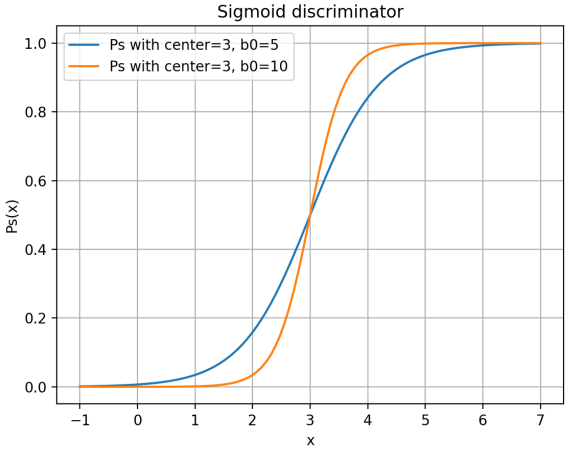


Figure 14: The P_s function centered on 3 with different stiffnesses

Let's suppose one has a measurement x that tells one something about an event A. This function gives one an interval $[x_0, x_1]$ where A is almost-impossible, and an interval $[x_2, x_3]$ where A is almost-certain.
This can be useful to maps values of torques or contact forces to a probability of contact.

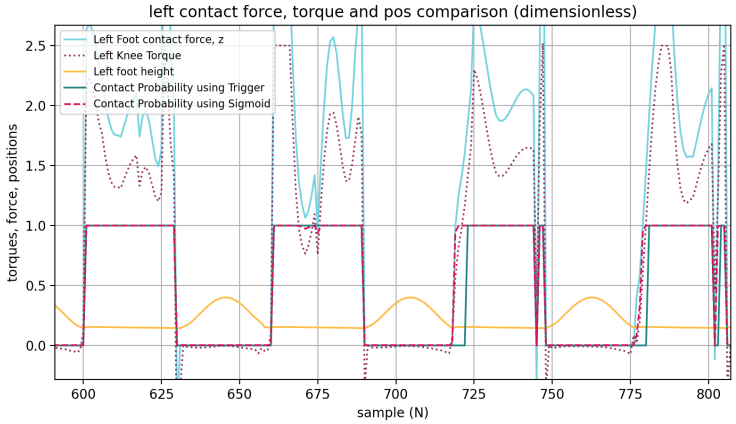


Figure 15: Using the P_s function to detect ground contact, using contact force.
It proves to be more precise and reactive than a detection using upper and lower trigger.

5.4 Slipping

For a contact force $F_i^{3D} = \begin{bmatrix} F_i^x \\ F_i^y \\ F_i^z \end{bmatrix}$ we consider the coefficient []

$$\mu = \frac{\sqrt{F_i^{x2} + F_i^{y2}}}{F_i^z}$$

Then, $\mu < \mu_{thresold}$ gives us a criteria about the foot's likeliness to slip. This coefficient is used in [10]. Other criterias are the foot's horizontal speed and the foot's horizontal acceleration. Each of these coefficients is composed by the P_s function to discriminate cases.

5.5 Detecting Switches

Switches denote the moment when the robot changes its stance foot. For instance, the switch can be when the left foot touches the ground, followed shortly afterward by the right foot moving up.
The robot can have both of its feet in contact with the ground for some time. This gait is called double support. It might also lift its foot at the exact same time the swing foot touches the ground. This gait is called simple support. In both case, the estimation of contact is not perfect especially around switch times, so there might be unwanted contact detection. To accurately detect the switches, we proceed as follow : To prevent a switch from lasting forever in case of a wrong initialization, a switch timeout is added to this algorithm.

6 Estimating Position

6.1 Goal

In order to compute the DCM, one needs the position and speed of the center of mass of the robot. These can be infered from the position and speed of the robot's base. We have estimated the speed in 4, we now need to estimate the position.

Algorithm 1 Basic switch detection

1. Initialize *ContactStatus* and robot state as not *Switching*.
2. While *True*:
   (a) If a change is detected in *ContactStatus* and robot is not *Switching*
   ↪ robot is now *Switching*
   (b) If robot is *Switching*
   ↪ do what needs to be done (such as computing feet position)
   (c) If two changes are detected in *ContactStatus*
   ↪ swing foot and stance foot have exchanged roles. Robot is no longer *Switching*.

6.2 Principle

Bolt is a blind robot : it does not have absolute odometry, through Lidar or vision for instance. Our first option is then to integrate the acceleration from the IMU two times :

$$p_{imu} = \iint_t a_{imu} \, \mathrm{d}t$$

But this is of course drifting very quickly because of noise in a_{imu} and numerical integration errors. It can only provide accurate position estimate for a very short period of time.

Another way is to integrate the previously estimated speed.

$$p_{inter} = \int_t v_{out} \, \mathrm{d}t$$

This will also drift after a few seconds, because of errors in estimate and numerical integration errors. Yet, it provides a much improved estimate of p.

A third way is to sum the length of a footstep. For a very short period of time, every time Bolt puts a new foot on the ground and lift the other, it has its two feet on the ground. We call that a *switch*. During this period, we assume both feet are at a standstill. The distance between the two feet in contact with the ground can then be computed thaks to the encoders data. We thus obtain the length of a footstep.

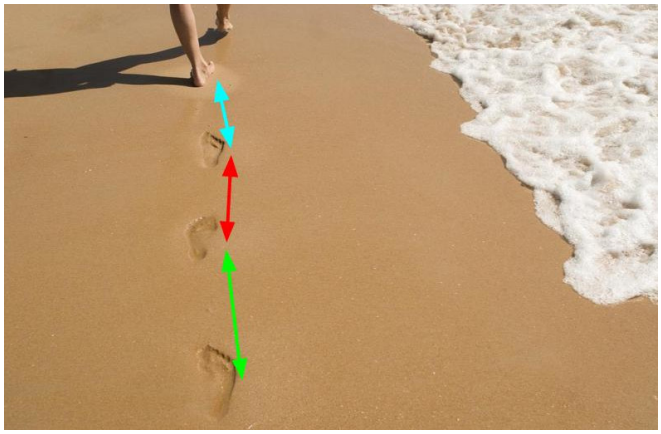


Figure 16: Switch odometry

One can see that adding the footstep lengths provides the position of the walker with regards

to its starting position. This can be used to compensate the drift of the previous methods, by providing a close estimate of the robot position every footstep.

Below is the result of switch odometry on a simulation of Bolt. Estimator runned at 1kHz for 4 seconds. One can see that footsteps and switches were acurately detected. The shift in y position is due to the alternance between left foot and right foot, and that could be corrected easily. The z estimate is drifting.

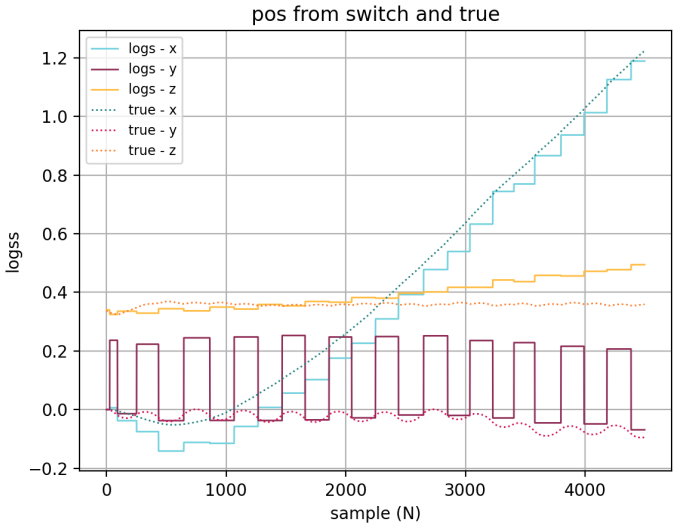


Figure 17: Switch odometry in Bolt simulation

6.3 Implementation

The position p_{out} is computed as follow : $\Delta p = p_{inter} - p_{switch}$ is computed every footstep. Then, during the following footstep, the error on x is compensated by adding $\frac{\Delta p}{\Delta T_{footstep}}$ to the p estimate. This way, assuming the drift is slow enough and the next footstep is about the same duration as the previous one, the drift will be compensated at the time of the next switch. Since the drift using

$$p_{inter} = \int_t v_{out}\,\mathrm{d}t$$

takes more than one second to become significant, the x position drift can be compensated using the switch odometry. As for the z drift on position, it is corrected using the encoders. The estimator computes the z distance between the foot in contact and the base every timestep, and use it to correct this drift.

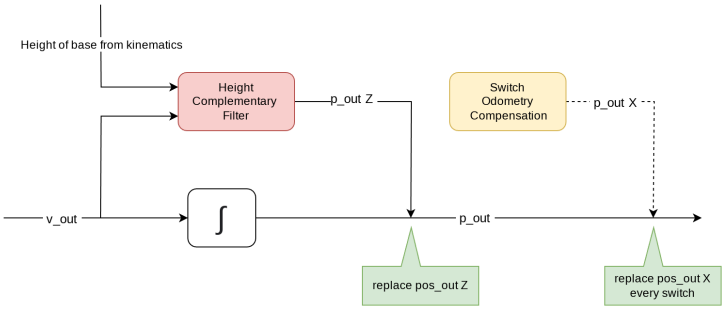


Figure 18: Position datapipe

7 Testing

7.1 Testing Filters

To test the filters, on can generate simple signals, add gaussian white noise to a certain intensity and feed it to the filter Below is the structure in place for testing the different filters.

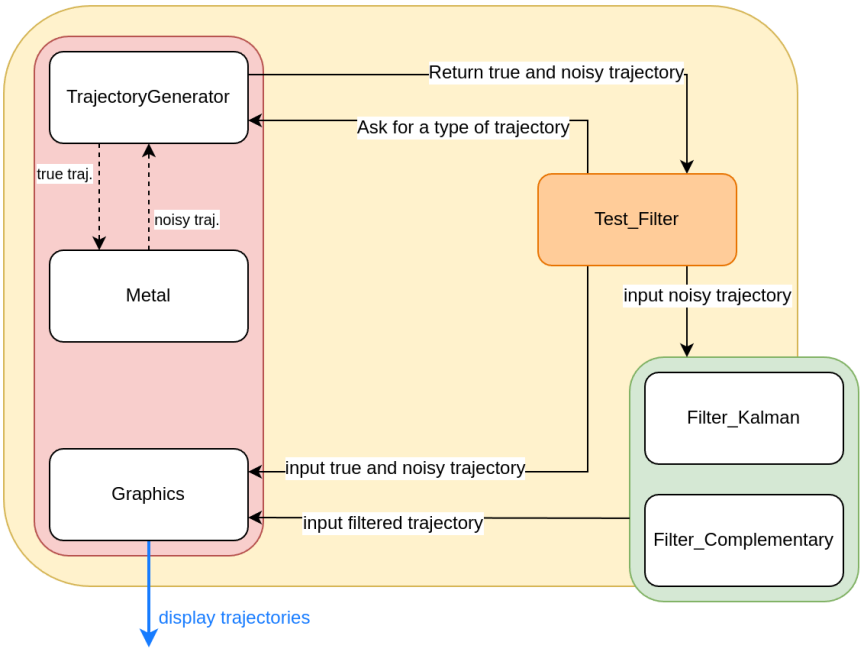


Figure 19: Filter testing on simulation logs

7.2 Testing Estimator on logs

To test the main estimator and its sub-estimators, one can save logs from a simulation, and then run the estimator on this dataset. This allow to log the full state of the robot, including data that the estimator is not able to read (such as the base position, or the exact movement of the hip). This data can then be used to compare the estimator output to the simulated truth, and to understand what is going on during estimation.

Below is the structure in place for testing the Estimator on dataset extracted from simulation. DeviceEmulator acts as replacement for the ODRI interface.

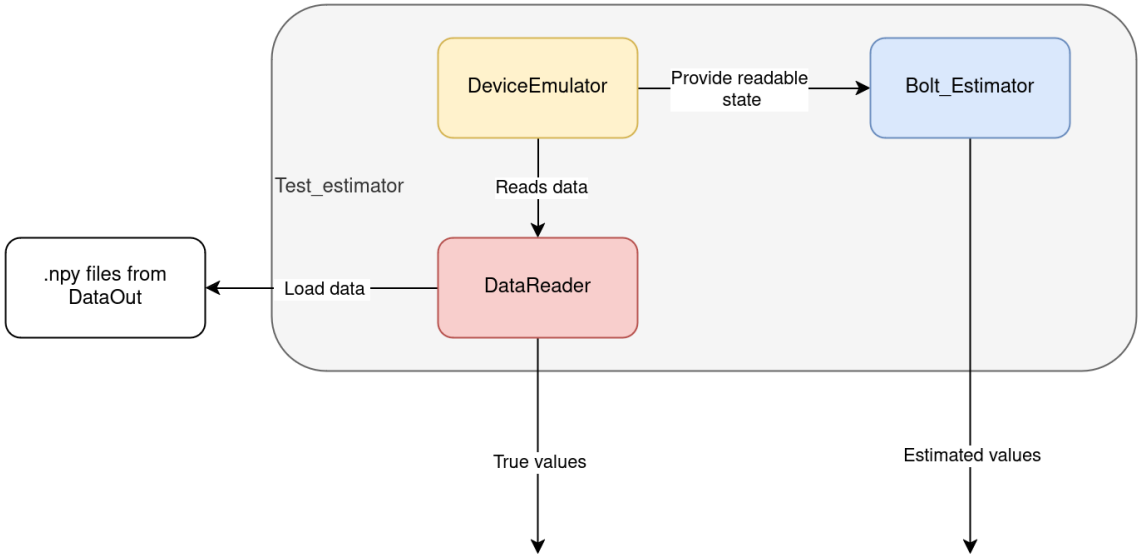


Figure 20: Estimator testing on logs

I chose to test the estimator on four different datasets, with different dt at 1ms (1kHz), 5ms (200Hz) and 10ms (100Hz). Below are representative results obtained after a run of 3s at 1 kHz.

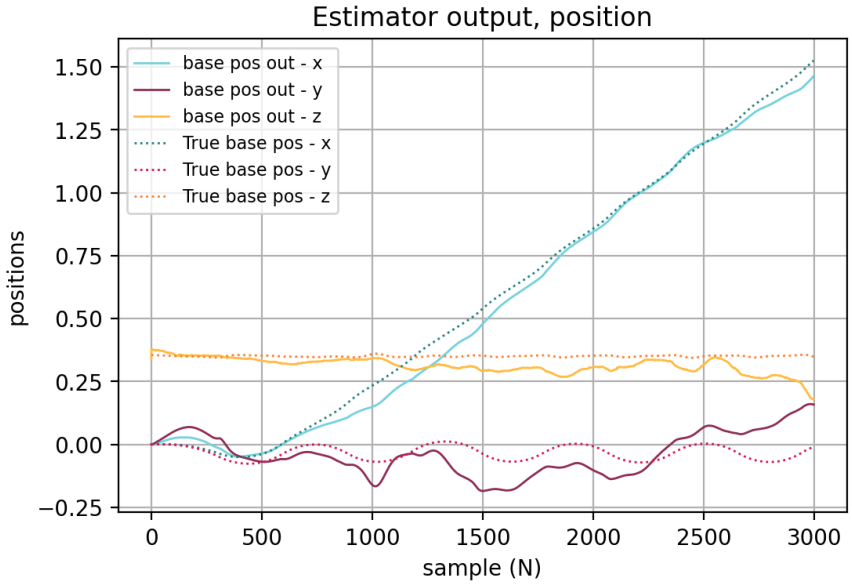


Figure 21: Position (with switch compensation starting at 1500ms)

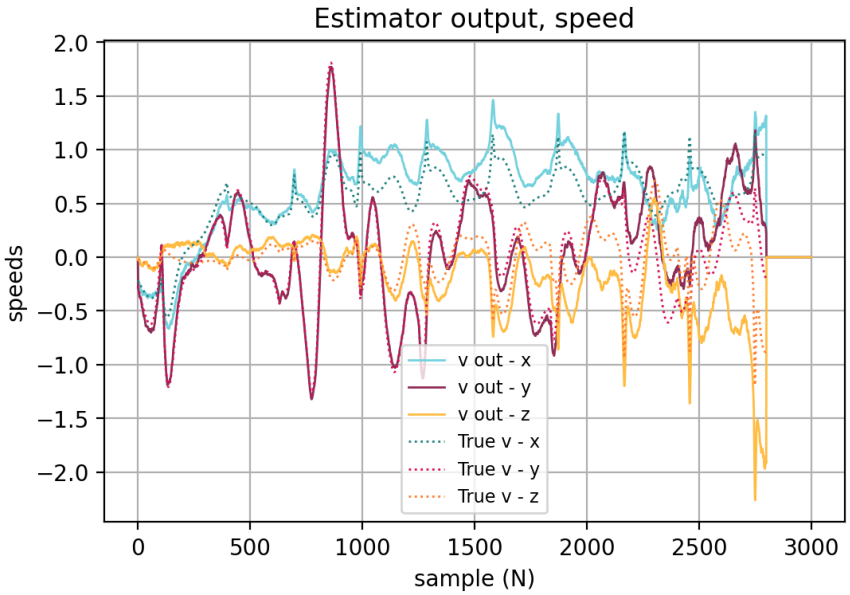


Figure 22: Speed

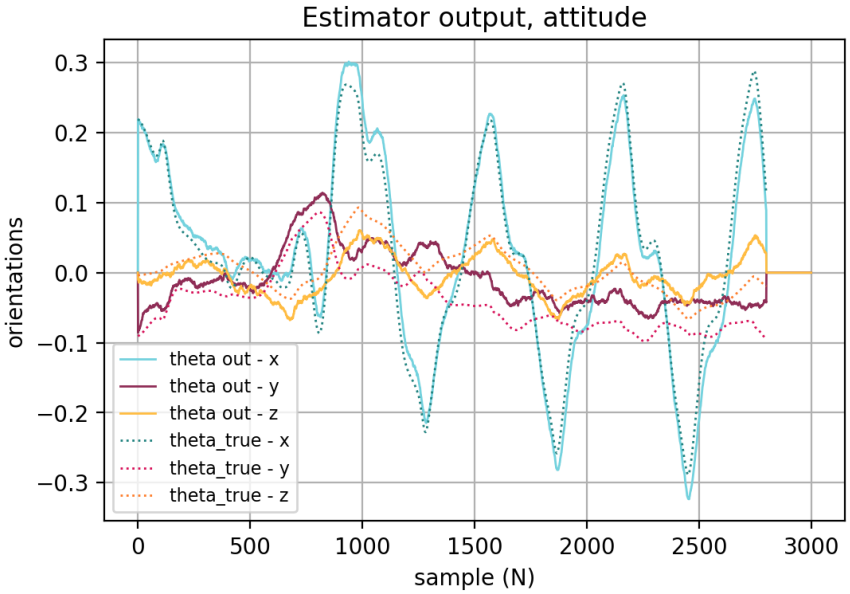


Figure 23: Attitude

7.3 Testing Estimator in live simulation

To further test the estimator, one can run it directly within the simulation. The estimator can be fed with true data from the solver. The estimated data it returns can be logged or used to compute the command. The true, command and estimated values can be plotted using DataLogReader.

8 Testing on Bolt

8.1 Testing wuth IMU and MoCap

9 Hardware modification

9.1 Goals

Bolt is an open-source project, meaning that all plans and information on hardware and software are freely available. This allows one to easily modify or re-design some parts of the robot. I had to do this for several reasons.

9.2 Wiring

We put the boards back in place and re-wired the robot with C.Roux, following instruction on [1].

9.3 Stand

A stand was made previously for Bolt. However, it was ill-suited for the robot, as Bolt was unstable when on it. PLus, its legs were touching the ground. I redesigned a new one, partly inspired by the photograph of Bolt stand available on [1]. However, I changed the dimensions and added several parts to it. The stand is made out of 4 aluminium profiles, 4 brackets, and 3 3D-printed parts.

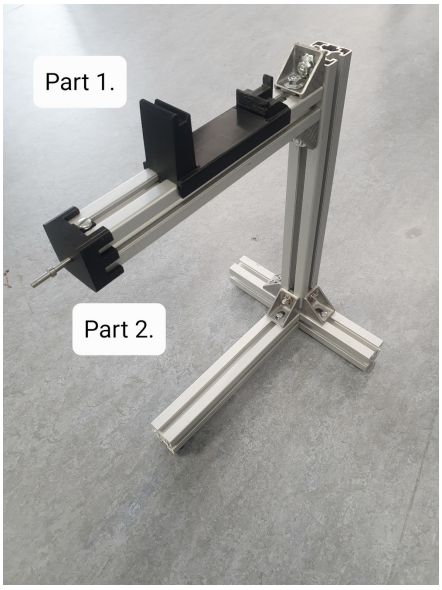


Figure 24: The new Bolt stand

Bolt can be placed either on top of the horizontal bar as in 25 or on the protruding axe. In the first case, its feet cannot touch the ground and its legs can move freely. The body is firmly held due to the V-shape of the front and back parts of Part 1. Part 1 can slide back and forth. This position is useful for calibrating the robot.

Figure 25: Bolt in position 1, with and without Part 3 on.

When Bolt is placed on the protruding axis, its base height is such that it is in its ideal starting position. Part 3 can bu put front of Part 2, or not. If it is, the robot will be held only by the very end of the axis. Its back leans against Part 3, ensuring it is in a vertical position and not leaning forward. This allows the robot to be in a good starting position for a walk, and detach itself from the stand at the very start of its movement. If Part 3 is not on, the robot will lean slightly forward, but will have a firmer attach to the stand, preventing it from falling easily.

The axis used is 5mm in diameter and around 56mm in length. Part 2 has a central extension inside the aluminium profile, to hold the axis strongly. The axis can be screwed to the back of Part 2. I recommend Part 2 to be printed with reasonnable quality and high strength settings, with an infill of around 35%. Dimensions of parts 2 and 3 were made for a specific aluminium profile, as their shapes closely match those of the profile. All parts were printed in PLA.

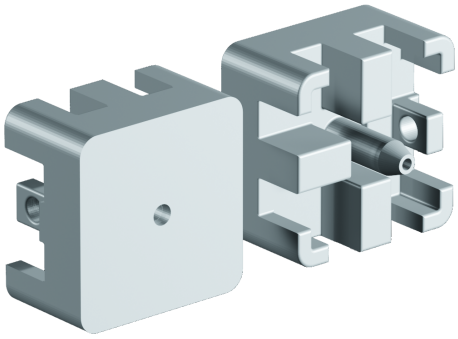
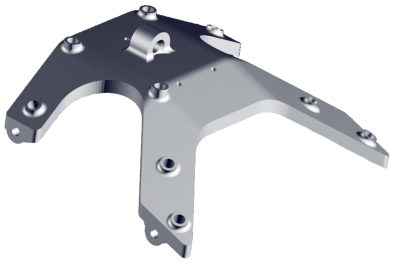


Figure 26: render of Part 2

9.4 Bolt's hat

The hat that was originally printed for Bolt has many supports for MoCap markers, but all are very close from one another. This proved to cause issues with the MoCap system at LAAS, probably because the marker used were bigger than those originally fitted. Additionnaly, the hat was not very strong. I designed and built a new hat, which features more dissymetric and distant MoCap markers mounting points, a stronger build, and a hook. The hook can be used to attach a cable to lift the robot, or hold it in the event of a fall. Due to the additionnal thickness, the hat protrudes upward slightly from Bolt's chassis. It took three iterations of this part to get the MoCap markers placement right.



(a) Render of the new hat

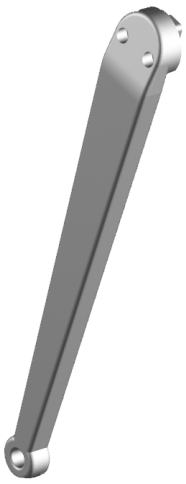
(b) The new Bolt hat and the MoCap markers we used

Figure 27: Bolt's new hat

Figure 28: Hooking Bolt for testing

9.5 Bolt's legs

Bolt's original legs have a distinctive passive actuator on the ankle, and a cylindric foot. To get closer to the URDF design and for testing purpose, I designed a new leg, with the same length and shape as the URDF's leg. In both case, the foot-ground contact is more or less a sphere-plane contact.



(a) Render of the new leg

(b) Leg with passive actuation in red, new leg with rounded end in black

Figure 29: Bolt's new leg

10 Context

This section formally introduces the organizations mentionned in this document.

10.1 What is CNRS ?

CNRS is the french abbreviation for National Scientific Research Center. CNRS is mainly funded by the French government, and has many compounds all over France. Founded in 1939, it is widely multidisciplinary and has strong international connections.
`https://www.cnrs.fr/en`

10.2 What is LAAS ?

LAAS is a research laboratory, part of CNRS. It focuses on computer science, robotics, automation, and nano-technologies. Located in Toulouse, S-W of France.
`https://www.laas.fr/en/`

10.3 What is Gepetto ?

Gepetto is a team of researchers, working at LAAS and specialized on legged robots.
`https://www.laas.fr/en/teams/gepetto/`

11 Explanations

This section will hopefully answers your questions over various aspects of this work. It defines several terms, as they are used in this document. They are not to be considered as formal definitions, but rather as a starting point for latter research if needs be. Moreover, the following notions are merely put in the context of this document. For more information, make use of the references linked to each of the following point.

11.1 What is Bolt ?

Bolt is a bipedal robot, around 40cm in height and 1.3kg heavy. It is an open-source project, whose CAD can be found here : `https://github.com/open-dynamic-robot-initiative/open_robot_actuator_hardware/blob/master/README.md`
It was developped under the Open Dynamics Robot Initiative. Bolt is the 2-legged version of Solo, which is 4-legged. Bolt has 6 DoF, 3 per leg. Its ankle are unactuated.

Figure 30: Bolt

11.2 What is Bolt's base ?

Bolt's base is what would be its pelvis, should Bolt have been human. In fact, given the physionomy of Bolt, it is anything but its legs.

11.3 What is an IMU ?

IMU stands for Inertial Measurement Unit. It is a set of sensors, including at least a 3D accelerometer and a 3D gyroscope, which gives us a 6-axis IMU, that can be extended with a magnetometer, which gives us a 9-axis IMU.

Figure 31: IMU

11.4 What is MoCap ?

MoCap is the abbreviation for Motion Capture, and refers to a system of n cameras being used to derive the position, attitude, speed, etc of a given object in space. For the cameras to understand where the object is, specific markers are added on known spots on the object.

Figure 32: Bolt's new hat with 4 MoCap markers

11.5 What is a DoF ?

DoF stands for degree of freedom. It is the number of parameter you can independantly change in a system. A leg with $k \in \mathbb{N}$ DoF has k controled joints that can be controled independantly.

11.6 What is Attitude ? What is Tilt ?

The attitude, or tilt, is the set of three angles our robot makes with a world reference. The robot might tilt left (roll), tilt nose-down (pitch), or circle around on the floor (yaw).
`https://en.wikipedia.org/wiki/Orientation_(geometry)`

11.7 What is an URDF ?

URDF is a file format that stands for Universal Robot Description Format. It contains the physical description of the robot, including dimensions, weight, and inertias. An URDF file describes all joints, and links them together in a tree-like fashion.

11.8 What is a Quaternion ?

In our situation, a (unitary) quaternion is a way to represent a 3D rotation. It can be seen as a vector representing the axis of the rotation, concatenated with the sinus of the angle about this axis. The most qualified person to explain this :
`https://www.youtube.com/watch?v=d4EgbgTm0Bg`

11.9 What is Solo ?

Solo is a quadruped robot, developped by the Open Robot Dynamic Initiative. It exists in 8 and 12 DoF, named Solo8 and Solo12.

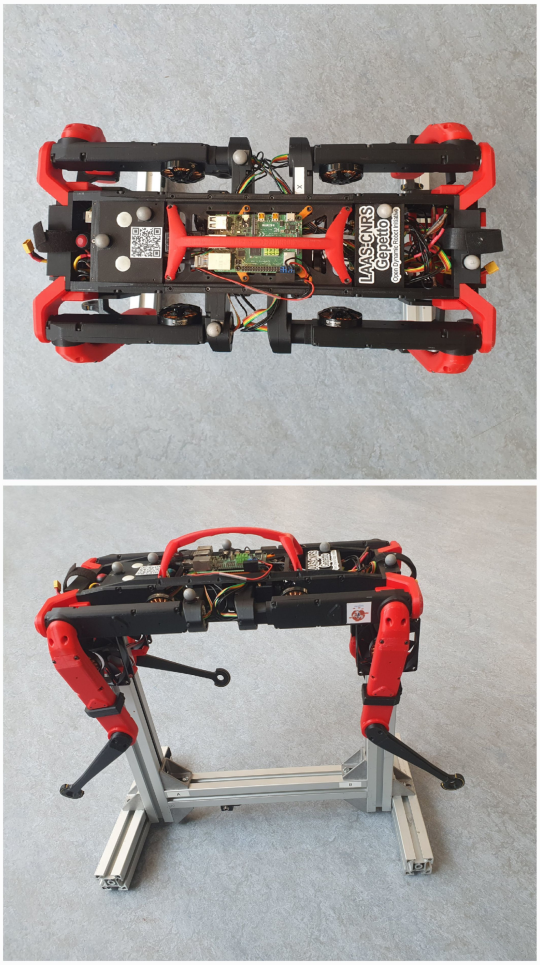


Figure 33: Solo 12

11.10 What is Pinocchio ?

Pinocchio is a library used for dynamical computations on robots. It has a large set of function for computing rotations, jacobians, forward kinematics, inverse kinematics, and so on. It is one of the, if not the one, most computationnally efficient library available for those tasks. From the user point of view, Pinocchio takes care of all the dirty physical equations in the background as long as you give it the right input. However, documentation is scarce. Version used was 2.7. More information can be found at `https://github.com/stack-of-tasks/pinocchio`

11.11 What is Croccodyl ?

11.12 What is ROS 2 ?

12 Bibliography

References

[1] open robot actuator hardware BOLT - github.com, https://github.com/open-dynamic-robot-initiative/open_robot_actuator_hardware/blob/master/mechanics/biped_6dof_v1/README.md.

[2] O. Stasse, Git Bolt Ros2, https://github.com/stack-of-tasks/odri_bolt_robot.

[3] L. Electronics, Bolt's IMU, https://www.microstrain.com/inertial-sensors/3dm-cx5-25.

[4] F Kung, Complementary filter, https://fkeng.blogspot.com/2018/05/digital-implementation-of-complementary.html.

[5] Y.-B. Jia, *Quaternions and Rotations*, **2013**.

[6] M. Benallegue, R. Cisneros, A. Benallegue, Y. Chitour, M. Morisawa, F. Kanehiro, *Lyapunov-stable orientation estimator for humanoid robots*, **2020**.

[7] M. Fourmy, T. Flayols, P.-A. Léziart, N. Mansard, J. Solà, *Contact Forces Preintegration for Estimation in Legged Robotics using Factor Graphs*, **2021**.

[8] E. Zurich, *Robot Dynamics Lecture Notes, pp.72-73*, **2017**.

[9] S. Yang, H. Kumar, Z. Gua, X. Z. et alt, *State Estimation for Legged Robots Using Contact-Centric Leg Odometry*, **2019**.

[10] M. Camurri, M. Fallon, S. Bazeille, A. Radulescu, V. Barasuol, D. G. Caldwell, C. Semini, *Probabilistic Contact Estimation and Impact Detection for State Estimation of Quadruped Robots*, **2018**.