Multisensory State Estimation and Mapping on Dynamic Legged Robots

Marco Camurri

Istituto Italiano di Tecnologia, Italy
Università degli studi di Genova, Italy

Thesis submitted for the degree of:
Doctor of Philosophy (PhD)
March 2017

Tutor:
Dr. Claudio Semini
Dynamic Legged System Lab
ISTITUTO ITALIANO DI TECNOLOGIA

Co-Tutor:
Dr. Stéphane Bazeille
DAPI / IRCCyN, Équipe Robotique
ÉCOLE NATIONALE SUPÉRIEURE DES MINES DE NANTES

Reviewers:
Dr. Giorgio Grisetti
Dept. of Systems and Computer Science
LA SAPIENZA UNIVERSITY OF ROME

Dr. Ludovic Righetti
Movement generation and control group
MAX-PLANCK-INSTITUTE FOR INTELLIGENT SYSTEMS

This dissertation was typeset with LaTeX on Linux.
Model adapted from "TesiModerna" by Lorenzo Pantieri (http://www.lorenzopantieri.net).

To my parents, Danilo and Franca,
and my fiancée, Claudia.

Contents

List of Figures ix

List of Tables x

1 Introduction 1

1.1 Contributions . . . 3

1.2 Outline . . . 3

2 Literature Review 5

2.1 Proprioceptive State Estimation . . . 6

2.1.1 Strapdown Inertial Navigation . . . 6

2.1.2 Early Proprioceptive Sensor Fusion . . . 9

2.1.3 Kinematics-Inertial State Estimation . . . 12

2.2 Contact Estimation Methods . . . 15

2.3 Exteroceptive Pose Estimation Methods . . . 15

2.3.1 Visual Odometry . . . 15

2.3.2 Visual SLAM . . . 17

2.3.3 Iterative Closest Point . . . 17

2.3.4 Normal Distribution Transform . . . 19

2.4 Multisensor State Estimation . . . 19

2.5 Mapping for Legged Robot Navigation . . . 20

2.5.1 Elevation Map . . . 21

2.5.2 Point Cloud . . . 22

2.5.3 OctoMap . . . 22

2.6 Summary . . . 22

3 System Overview 25

3.1 Mechanical Design . . . 25

3.2 Frames of Reference . . . 25

3.3 Proprioceptive Sensors . . . 26

3.3.1 Encoders . . . 26

3.3.2 Joint Force/Torque Sensor . . . 28

3.3.3 Inertial Measurement Unit . . . 28
3.4 Exteroceptive Sensors . . . 28
3.4.1 Depth Sensor . . . 28
3.4.2 Stereo Camera . . . 29
3.4.3 LiDAR . . . 30
3.4.4 Depth Sensor Noise and Calibration . . . 30
3.5 Sensor Configurations . . . 30
3.5.1 Configuration A . . . 31
3.5.2 Configuration B . . . 32
3.6 Camera Pose Calibration . . . 33
3.7 Synchronization . . . 34
3.8 Locomotion and Perception Capabilities . . . 35

4 Probabilistic Contact Estimation for Proprioceptive State Estimation 37
4.1 Filter Framework . . . 38
4.1.1 Inertial Process Model . . . 38
4.1.2 Measurement Model . . . 40
4.2 Probabilistic Contact Estimation . . . 40
4.2.1 Ground Reaction Forces Estimation . . . 41
4.2.2 Contact Classification . . . 42
4.2.3 Training Set Generation . . . 42
4.2.4 Performance Evaluation . . . 45
4.3 Measurement Integration . . . 47
4.3.1 Velocity Estimation . . . 48
4.4 Covariance Estimation . . . 49
4.4.1 Inter-Leg Variance . . . 49
4.4.2 Impact Detection . . . 50
4.5 Experimental Results . . . 51
4.5.1 System Overview . . . 51
4.5.2 Performance Evaluation . . . 52
4.6 Discussion . . . 56
4.6.1 Leg Compliance . . . 56
4.6.2 Limitations . . . 57
4.7 Conclusion . . . 57

5 Multisensor Fusion for Accurate Pose Estimation 59
5.1 Requirements . . . 61
5.2 Selective Iterative Closest Point . . . 61
5.2.1 Method Description . . . 62
5.2.2 Inertial Measurement Integration . . . 64
5.2.3 Experimental Results . . . 66

5.2.4 Discussion . . . 66
5.3 Fast and Robust Scan Matcher . . . 68
5.3.1 Experimental Results . . . 69
5.4 Gaussian Particle Filter . . . 72
5.4.1 Scan Filtering . . . 72
5.4.2 Experimental Results . . . 72
5.5 Multisensory State Estimation . . . 74
5.5.1 Visual Odometry . . . 75
5.5.2 Auto-tuned ICP . . . 76
5.5.3 Fusion Algorithm . . . 76
5.5.4 Experimental Results . . . 77
5.6 Conclusion . . . 80

6 Robocentric Mapping and Locomotion Applications 83
6.1 Mapping Representations . . . 84
6.1.1 Global Mapping vs. Robocentric Mapping . . . 84
6.1.2 Robocentric Point Cloud Mapping . . . 86
6.1.3 Elevation Mapping . . . 88
6.2 Terrain Pattern Classification for Visual Reactive Trotting . . . 89
6.2.1 Foothold Correction as a Classification Problem . . . 89
6.2.2 Training Set Generation . . . 90
6.2.3 Logistic Regression for Foothold Decision . . . 90
6.2.4 Experimental Results . . . 91
6.3 Terrain Cost Map for Foothold Planning . . . 92
6.4 Discussion . . . 93
6.5 Conclusion . . . 94

7 Conclusion and Future Work 95

Appendices 97

A Datasets 99
A.1 Dataset 1 . . . 99
A.2 Dataset 2 . . . 100
A.3 Dataset 3 . . . 102

B Gaussian Filters 105
B.1 Kalman Filter . . . 105
B.2 Extended Kalman Filter . . . 106
B.3 Unscented Kalman Filter . . . 107
B.4 Gaussian Particle Filter . . . 108

C Performance Metrics **111**

C.1 Drift per Distance Traveled 111

C.2 Root Mean Square Error 112

Bibliography **113**

List of Figures

2.1 Robots used in the early stages of leg odometry . . . 11
2.2 Examples of quadruped robots . . . 13
2.3 Examples of humanoid robots . . . 13

3.1 Joints names and planes conventions on HyQ . . . 26
3.2 The Hydraulic Quadruped robot and its frames of reference . . . 27
3.3 Proprioceptive sensors mounted on HyQ . . . 27
3.4 Exteroceptive sensors mounted on HyQ . . . 29
3.5 Exteroceptive sensors in Configuration A . . . 32
3.6 Exteroceptive sensors in Configuration B . . . 33
3.7 Sensor calibration setup . . . 34

4.1 Crawl gait simulation . . . 45
4.2 Trot gait simulation . . . 46
4.3 Crawl gait experiment . . . 47
4.4 Trot gait experiment . . . 48
4.5 Velocity comparison with different contact estimation methods . . . 49
4.6 Effect of impulsive force on estimated velocities . . . 51
4.7 Trotting raw velocity compared to ground truth . . . 52
4.8 Schematic of the EKF framework used on HyQ. . . . 53
4.9 Position comparison between baseline and proposed method . . . 54
4.10 Example of velocity estimation for a trot log on x-axis . . . 54
4.11 Drift per Distance Traveled (DDT) for trot and crawl logs . . . 55
4.12 Root Mean Squared Error (RMSE) for trot and crawl logs . . . 55
4.13 HAA Torque vs. Position curves of legs under load/unload . . . 56

5.1 Image preprocessing for point selection . . . 63
5.2 Selective ICP vs. ICP . . . 64
5.3 Example of online mapping with the Selective ICP algorithm . . . 66
5.4 Experiment with the real robot trotting indoor . . . 67
5.5 Average DDT on the trot and crawl logs taken from Dataset 2 . . . 69
5.6 Position of IMU + LO + FRSM modules against Vicon . . . 70

5.7 Orientation of IMU + LO + FRSM modules against Vicon . . . 70
5.8 Push recovery with state estimation in the loop . . . 71
5.9 Comparison between unfiltered and filtered point clouds . . . 73
5.10 Drift per Distance Traveled on trot and crawl logs from Dataset 1 . . . 73
5.11 Position of IMU + LO + GPF modules against Vicon . . . 73
5.12 Orientation of IMU + LO + GPF modules against Vicon . . . 74
5.13 Example of concurrent pose correction from LiDAR and Visual Odometry. . . . 76
5.14 Estimated trajectories with different sensor modalities . . . 78
5.15 Pose Estimation in-the-loop: indoor square wave experiment . . . 78
5.16 Pose Estimation in-the-loop: outdoor square wave experiment . . . 79
5.17 Example of online LiDAR mapping with multi-sensor state estimation . . . 79

6.1 Example of local mapping . . . 86
6.2 Robocentric map example . . . 87
6.3 Local heightmap example . . . 88
6.4 Point cloud and heightmap . . . 90
6.5 Examples of patterns, taken from the training set . . . 90
6.6 Log traversal experiment . . . 91
6.7 Foothold planning results using a heightmap . . . 92

A.1 Typical crawl and trot base trajectories . . . 101
A.2 Cropped camera and depth images from Multisense SL . . . 103
A.3 Detail of the Multisense SL protection frame . . . 103
A.4 Test area for Dataset 3 . . . 103

List of Tables

2.1 Comparison of different state estimation approaches. . . . 23

3.1 Comparison between depth sensors specification used on HyQ. . . . 31
3.2 List of sensors and additional actuators. . . . 31

4.1 DDT of different contact estimators . . . 47

5.1 Limitations of the exteroceptive pose estimation methods. 80

A.1 Summary of Dataset 1. 100
A.2 Summary of Dataset 2. 101
A.3 Summary of Dataset 3 102

Acronyms

AICP Auto-tuned Iterative Closest Point. 19, 74, 75, 78

COM Center Of Mass. 14

DDT Drift per Distance Traveled. 43, 50, 67

DoF Degree of Freedom. 6, 10, 14, 23, 39, 57, 64, 65, 70

DRC DARPA Robotics Challenge. 17, 20, 26

EKF Extended Kalman Filter. 10, 12, 14, 18, 19, 36–38, 45, 49, 57, 58, 66, 68, 70–74, 83, 91, 102, 103

FOG Fiber Optic Gyroscope. 6, 26, 30

FoV Field of View. 28–31, 58, 60, 72, 74, 78, 83

FRSM Fast and Robust Scan Matcher. 58, 64, 66, 67, 70, 72, 78

GPF Gaussian Particle Filter. 19, 58, 70, 72, 74, 78

GPS Global Positioning System. 9, 10, 21

GRF Ground Reaction Force. 35, 39–41, 48

HAA Hip Abduction-Adduction. 23, 26, 28, 54

HFE Hip Flexion-Extension. 23, 26

HyQ Hydraulic Quadruped. 21–24, 26–29, 32, 33, 35, 36, 38, 39, 54, 57–59, 61, 64, 66, 70, 72, 75, 78–80, 82, 87, 91, 95

ICP Iterative Closest Point. 17, 18, 20, 58–63, 66, 72–74, 78, 80, 82

IF Information Filter. 19

IMU Inertial Measurement Unit. 7, 9, 10, 14, 18, 19, 26, 30, 33, 35–37, 62–64, 66, 67, 70, 73, 75, 95, 98

IR Infrared. 26, 27, 60

KF Kalman Filter. 10, 14, 19

KFE Knee Flexion-Extension. 23, 26

LCM Lightweight Communication and Marshalling. 33, 95

LF Left Front. 23, 84

LH Left Hind. 23

LiDAR Light Detection And Ranging. 2, 5, 19, 20, 26–28, 30, 32, 58, 73, 74, 77, 80, 82, 92, 95

LIPM Linear Inverted Pendulum Model. 14

LO Leg Odometry. 9, 10, 15, 19, 36, 45, 66, 67, 70, 72, 73, 75, 91, 95

MEMS Micro Electro Mechanical System. 6, 10, 18, 26

QP Quadratic Programming. 14

RCF Reactive Controller Framework. 33, 85

RF Right Front. 23

RH Right Hind. 23

SAD Sum of Absolute Distance. 16, 73

SLAM Simultaneous Localization and Mapping. 17, 18, 79, 92

SLIP Spring Loaded Inverted Pendulum. 14

SVD Singular Value Decomposition. 9, 32

UKF Unscented Kalman Filter. 12, 14, 103

V-SLAM Visual SLAM. 17

VO Visual Odometry. 15–17, 19, 31, 58, 73–75, 78, 92

Notation

Sets

$\mathbb{R}$	Set of real numbers
$\mathbb{B}$	Set of binary numbers
$\mathbb{N}$	Set of natural numbers

Vectors and Matrices

$x,\ \alpha$	scalars
$\mathbf{x},\ \boldsymbol{\alpha}$	vectors
${}_a\mathbf{x}_b$	vector referred to frame b expressed in frame a
${}_a\mathrm{T}_b$	transformation from frame b to frame a such that ${}_a\mathbf{p} = {}_a\mathrm{T}_{b\,b}\mathbf{p}$
A, B	matrices
A^{T}	matrix transpose
A^{-1}	matrix inverse
$\mathbf{x}^{\wedge} = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}$	skew symmetric matrix of $\mathbf{x} \in \mathbb{R}^3$
$\Lambda(\mathbf{x}) = \begin{bmatrix} x_1 & 0 & 0 \\ 0 & x_2 & 0 \\ 0 & 0 & x_3 \end{bmatrix}$	diagonal matrix of $\mathbf{x} \in \mathbb{R}^3$

Other symbols

$\boldsymbol{x}$	state vector (Kalman)
$\boldsymbol{u}$	input vector (Kalman)
$\boldsymbol{z}$	measurement vector (Kalman)
$\mathcal{N}(\boldsymbol{\mu}, \Sigma)$	Gaussian with mean $\boldsymbol{\mu}$ and covariance Σ
$\mathcal{C}$	point cloud

Abstract

Legged robots are expected to outperform wheeled and tracked systems on unstructured, uneven, and rough terrains. However, they are characterized by higher mechanical complexity and they require sophisticated control strategies to maintain balance, generate appropriate feet and body trajectories, and navigate without falling, slipping or getting stuck.

The difficulty in the development of autonomous legged navigation lies in the close relationship between perception and locomotion. So far, research has been focusing primarily on locomotion, but in the near future the role of perception will increase in importance because it is crucial to operate outside the laboratory.

The objective of this dissertation is the introduction of the essential algorithms and tools required to achieve true autonomy (in terms of perception) and reduce the gap between perception and locomotion. In particular, it will focus on perception algorithms for semi-autonomous navigation of the agile, dynamic quadruped robot HyQ [131].

Throughout the dissertation, I will describe the algorithms and the methods to achieve accurate position and velocity estimates for perception-aware locomotion, as well as local mapping for visual obstacle negotiation. This thesis introduces a number of novel contributions which advance the state of the art in legged robot navigation, including: a new contact estimation method for state estimation; a new variant of the Iterative Closest Point algorithm for efficient localization; a multisensor approach to state estimation for smooth and accurate pose and velocity estimates in challenging conditions; and a local mapping classification method for visual reactive trotting.

We shall not cease from exploration
And the end of all our exploring
Will be to arrive where we started
And know the place for the first time.

— T.S. Eliot

Acknowledgments

This dissertation is the attempt to summarize the many days and nights spent at studying, coding, debugging, experimenting, breaking and repairing one of the most advanced quadruped robots available in research. These activities would not have been as exciting without the people who constantly encouraged and accompanied me in these three and a half years of PhD. With this small note I try to give them the credit they deserve.

My first thought goes to my family, who supported and loved me in every life choice I have taken. Danilo, thank you for showing me the courage to accept new challenges and the desire for new adventures. Franca, thank you for being my greatest example of patience and faith. Andrea, thanks for fostering my passion for science and technology.

Thanks to my friends from Modena: Daniele, Sara, Massimo, Giuseppe, Davide, Lorenzo, Teresa, Lucia, Alice, Sara and all the people who made me enjoy my undergrad years. Thanks to the people from the ImageLab for introducing me to the fascinating world of Computer Vision and for the opportunity to take my first steps in robotics.

Thanks to all past and present colleagues and friends of the Dynamic Legged Systems Lab at IIT: Stéphane, for investing on and tutoring me; Claudio, for supporting my professional growth; the PostDocs Marco, Michele, Victor, Andreea, Ioannis, for all the help and fun in and out the lab; the past and present PhD fellows Carlos, Romeo, Yifu, Octavio, Bilal, Hamza. Thanks to Roy Featherstone for his precious help in solving mathematical issues. Thanks to Prof. Caldwell for sharing his precious experience and supporting the HyQ project.

Thanks to our technicians: Jake, for your creative contribution in making HyQ2Max; Felipe and José, for all the software troubles we have solved together; Alex, for the help in solving big and small mechanical issues, and for organizing basketball games. Special thanks to Lisa, for proofreading this document.

Thanks to the ADVR technical staff: Riccardo Sepe, Marco Migliorini, Stefano Cordasco, Lorenzo Baccelliere, Gianluca Pane, Giuseppe Sofia, Alessio Margan, Francesco Di Dea, Paolo Guria, Phil Hudson.

Thanks to the secretaries for their support in the neverending battle against bureaucracy: Silvia Ivaldi, Giulia Persano, Elisa Repetti, Valentina Rosso, Floriana Sardi, Monica Vasco, Simona Ventriglia.

Thanks to Dr. Maurice Fallon for hosting me in Edinburgh and for his fundamental support, patience and guidance: *Go raibh maith agat!* Thanks also to his group in Edinburgh: Simona Nobili, Marco Caravagna, Wolfgang Merkt, Yiming Yang, Vladimir Ivan, and Prof. Sethu Vijayakumar.

Thank you Claudia, for your restless support and love.

Genoa, March 2017 M. C.

Ringraziamenti

Questa tesi è il tentativo di riassumere i numerosi giorni e le numerose notti spese a studiare, programmare, revisionare, sperimentare, rompere e aggiustare uno dei più avanzati robot quadrupedi disponibili per la ricerca. Queste attività non sarebbero state così emozionanti senza le persone che mi hanno costantemente incoraggiato ed accompagnato in questi tre anni e mezzo di dottorato. Con questa piccola nota cerco di dare loro il credito che meritano.

Il mio primo pensiero va alla mia famiglia, che mi ha sostenuto ed amato in ogni scelta di vita che ho preso. Danilo, grazie per avermi mostrato il coraggio di accettare nuove sfide e il desiderio di intraprendere nuove avventure. Franca, grazie per essere il mio più grande esempio di pazienza e fede. Andrea, grazie per aver alimentato la mia passione per la scienza e la tecnologia.

Grazie agli amici di Modena: Daniele, Sara, Massimo, Giuseppe, Davide, Lorenzo, Teresa, Lucia, Alice, Sara e tutte le persone con cui ho potuto godermi gli anni dell'Università. Grazie ai membri dell'ImageLab per avermi introdotto nell'affascinante mondo della visione artificiale e avermi dato l'opportunità di fare i primi passi nella robotica.

Grazie a tutti i colleghi (passati e presenti) e gli amici del Dynamic Legged System Lab in IIT: Stephane, per aver investito su di me e avermi seguito; Claudio, per avermi sostenuto nella mia crescita professionale; i PostDoc Marco, Michele, Victor, Ioannis, Andreea, per tutto l'aiuto e il divertimento dentro e fuori il laboratorio; grazie agli altri dottorandi passati e presenti: Carlos, Romeo, Yifu, Octavio, Bilal, Hamza. Grazie a Roy Featherstone per il suo aiuto prezioso nel risolvere problemi matematici. Grazie al Prof. Caldwell per aver condiviso con noi la sua preziosa esperienza e per il sostegno al progetto HyQ.

Grazie ai nostri tecnici: Jake, per la tua creatività nel costruire HyQ2Max; Felipe e José, per

tutti i problemi softwe che abbiamo risolto insieme; Alex, per l'aiuto nel risolvere problemi meccanici piccoli e grandi, e per organizzare le partite di basket. Un rigraziamento speciale a Lisa, per aver revisionato questo documento.

Grazie allo staff tecnico ADVR: Riccardo Sepe, Marco Migliorini, Stefano Cordasco, Lorenzo Baccelliere, Gianluca Pane, Giuseppe Sofia, Alessio Margan, Francesco Di Dea, Paolo Guria, Phil Hudson.

Grazie alle segretarie per il loro supporto nell'eterna battaglia contro la burocrazia: Silvia Ivaldi, Giulia Persano, Elisa Repetti, Valentina Rosso, Floriana Sardi, Monica Vasco, Simona Ventriglia.

Grazie al Dr. Maurice Fallon per avermi ospitato in Edimburgo e per il suo fondamentale aiuto, pazienza e assistenza: *Go raibh maith agat!* Grazie anche al suo gruppo di Edimburgo: Simona Nobili, Marco Caravagna, Wolfgang Merkt, Yiming Yang, Vladimir Ivan, and Prof. Sethu Vijayakumar.

Claudia, grazie per il tuo infaticabile sostegno ed amore.

Genova, Marzo 2017 M. C.

Chapter 1

Introduction

Legged robots are expected to outperform wheeled and tracked systems when they come to navigate in unstructured, uneven, and rough terrains. The superiority of legs over wheels comes from the fact that wheels require to be continuously in contact with the ground. This condition is rare in nature and has to be artificially created by building roads and rails. In contrast, legged locomotion is characterized by intermittent contact with the ground. This is a key advantage in terms of adaptability to challenging grounds, because the quality of the terrain outside the contact points is irrelevant, as long as enough leg clearance is guaranteed [135]. However, this advantage comes at a price: legged robots are characterized by high mechanical complexity. Furthermore, they require sophisticated control strategies to maintain balance and generate appropriate feet and body trajectories, to navigate without falling, slipping or getting stuck. To be competitive, a legged robot has to show these abilities in places which are not accessible by wheeled or tracked vehicles. These can include disaster zones, dense forests, and steep mountain trails. Even though legged robots have been developed for decades, the transition from laboratories to these real scenarios is still far from being complete.

The difficulty in the development of autonomous legged navigation lies in the close relationship between perception and locomotion. On one hand, the robot has to elaborate a myriad of signals (coming from itself and the environment) and combine them into useful information. On the other hand, it has to use this information to move in the environment. Furthermore, these two processes have to be concurrently executed onboard and in real-time. So far, the most logical approach has been focusing primarily on locomotion, with the role of perception limited to the minimum required to complete a specific task, or replaced by surrogate tools (*e.g.,* motion capture systems). However, in the near future the role of perception will increase in importance because it is essential to operate outside the laboratory. Autonomous robots will need to know their location, estimate their velocity, scan the environment, decide where to step, and where to go.

The objective of this dissertation is the introduction of the essential algorithms and tools required to achieve true autonomy (in terms of perception) and reduce the gap between perception and locomotion. In particular, it will focus on perception algorithms for semi-autonomous

navigation of the agile, dynamic quadruped robot HyQ [131].

There are at least three crucial capabilities a dynamic legged robot has to possess in order to navigate in a real scenario. These can be summarized as follows:

State estimation: how to compute (efficiently, robustly and at high frequency) an estimate of position, orientation, linear and angular velocity of the robot's base. When executed with only proprioceptive inputs, drift in position is unbounded and additional exteroceptive sources are required to accommodate for it.

Accurate pose estimation: how to keep the pose drift limited, by means of exteroceptive sensors like cameras, Light Detection And Ranging (LiDAR), or depth sensors.

Mapping: how to acquire and merge geometrical information about the environment in a compact, consistent and useful representation.

In this dissertation, I will cover these three critical aspects and address the inherent challenges of developing a perception system on a real robot. The tight coupling between perception and locomotion imposes stringent requirements in terms of frequency, accuracy, and smoothness of the estimates, as well as the resolution of the terrain maps. A typical control loop on HyQ operates at $1\,\mathrm{kHz}$ for the low level control and at $250\,\mathrm{Hz}$ for high level control (torque and trajectory generation). This implies that, to be safely used, the state estimator has to provide an output at the same (or higher) frequency of the high level control. At the same time, the small feet (4–$5\,\mathrm{cm}$ diameter) and the robot length ($1\,\mathrm{m}$) impose limits on the accuracy of the state estimator, which should not drift by more than one foot size per robot length (*i.e.,* $5\,\mathrm{cm/m}$ or $5\,\%$), in order to successfully avoid obstacles, edges, pitfalls and other potential hazards. Furthermore, the integration of pose and velocity estimates within the control loop requires them to be smooth, to avoid instabilities. Finally, the resolution of the map should capture the same level of detail required to properly place the feet on the ground. In our case, the resolution should be on a scale of $\sim 2\,\mathrm{cm}$ (half of the foot size). In addition to these requirements, other challenges include (but are not limited to): motion blur, scarce illumination, sensor pose calibration, signal synchronization, onboard computation, shocks, sharp turns, leg compliance, and two-way communication with the control loop.

Throughout the dissertation, I will describe the algorithms and the methods to achieve the following objectives:

- accurate and smooth velocity estimates for posture control and disturbance rejection during dynamic motions;
- accurate pose estimation on rough terrain for pose tracking and accurate mapping;
- robocentric local mapping and classification for visual reactive obstacle negotiation.

1.1 Contributions

This thesis describes a number of novel contributions which advance the state of the art in legged robot navigation, including:

- a contact estimation method for state estimation (published in [27] as first author);
- a new variant of the Iterative Closest Point algorithm, for efficient robot localization with depth sensors (published in [26] as first author);
- a multi-sensor approach to state estimation which fuses multiple proprioceptive and exteroceptive sources to attain smooth and accurate pose and velocity estimates, in challenging conditions (under review);
- a local mapping classification method to reactive trotting (published in [9] as second author).

All of the above works have been extensively validated on a variety of experiments carried out on the dynamic quadruped robot HyQ.

1.2 Outline

The rest of this dissertation is structured as follows: Chapter 2 describes the state of the art in state estimation, pose estimation and mapping in the context of legged robots; Chapter 3 gives an overview of HyQ, the research platform subject of the study presented herein, with specific focus to its sensory system, sensor noise characterization, and calibration; Chapter 4 describes the state estimation framework developed for HyQ, with particular focus to a probabilistic method for contact estimation suitable for Leg Odometry without dedicated contact sensors. The presented algorithm has been validated on over an hour of experiments; Chapter 5 is focused on pose estimation methods for agile dynamic quadruped robots. A new ICP-based algorithm is introduced, as well as a performance study of other state-of-the-art localization methods and a new multi-sensor approach to state estimation; Chapter 6 describes the mapping algorithms which have been successfully used in the robot locomotion control loop, including a pattern classification method for perception-driven reactive trotting; Chapter 7 concludes the dissertation with final remarks and further research prospects.

Chapter 2

Literature Review

This chapter presents the literature concerning the three key aspects of the navigation of legged robots: state estimation, pose estimation, and mapping. A brief digression on contact estimation is also given, since it is relevant for what will be presented later in Chapter 4.

Our review starts by discussing *proprioceptive* state estimation. The word *proprioceptive* indicates that the sensors used for the estimation are sensitive to physical quantities concerning the internal state of the robot, rather than the mutual relationship (*e.g.,* the distance) between the robot and the environment. Examples of proprioceptive sensors are: accelerometers, gyroscopes, encoders, and loadcells. It comes natural to start from proprioceptive sensors to estimate the robot's state. In this way, the estimation process is independent of the environment, which could be completely unknown, variable, and difficult to model.

Exteroceptive state estimation, on the other hand, involves the use of sensors which measure physical quantities related to the environment. Some examples are cameras, which measure the intensity of light reflected by an object, or LiDARs, which measures the distance to a target by illuminating it with a laser beam. Exteroceptive sensors are typically more complex and difficult to handle than proprioceptive ones. For instance, a single image from a 0.3 Megapixel monochromatic camera contains more than 300 kilobytes of data. In comparison, a single acceleration measurement can be expressed with just three floating point numbers, for a total of 12 bytes. Despite their complexity, exteroceptive sensors are fundamental to properly estimate the *pose* (*i.e.,* the position and orientation) of a robot. As a matter of fact, linear position and absolute yaw are quantities not observable by a proprioceptive estimator, which is affected by unbounded drift over these states. On the other hand, proprioceptive sensors are suitable for estimation of acceleration and velocity. Since legged navigation and locomotion require a reliable estimate of all the states mentioned above, the best results can be achieved by fusing both proprioceptive and exteroceptive sources into one estimate, typically in a Kalman filtering fashion, as we will see throughout the chapter.

Finally, to successfully navigate in difficult environments, a robot not only has to know its own state, but it also requires a good representation of its surroundings. This is important to plan body trajectories, select footholds, and promptly react to obstacles or other potential

hazards. Depending on the application, several world representations have been adopted in the past decades, ranging from simple 2D geometrical maps, to 3D meshes, and even to semantically enriched topological maps. Throughout the chapter we describe the mapping techniques which have been successfully adopted for locomotion and navigation of legged robots and mobile robots in general.

The rest of the chapter is structured as follows: Section 2.1 reviews the work related to proprioceptive state estimation, including inertial navigation, leg odometry, and the fusion of the two. Section 2.2 briefly introduces the topic of contact estimation. Although this is an interesting topic *per se*, we discuss it from the perspective of state estimation. In Section 2.3, we introduce the state-of-the-art techniques for pose estimation with exteroceptive sensors. We then continue with integrated approaches, which combine multiple modalities (inertial, kinematics, camera, and laser) to achieve better results. Section 2.5 reviews different mapping techniques and terrain representations for outdoor mobile robots. We also include some applications of mapping, such as terrain classification. Section 2.6 concludes the chapter with a summary of the previous sections.

2.1 Proprioceptive State Estimation

In contrast to industrial manipulators, legged robots have a floating base. This difference can be modeled by adding a virtual, unactuated joint with 6 Degrees of Freedom (DoFs) connecting the robot to an arbitrary coordinate frame fixed to the ground [40]. The goal of state estimation is to observe the state (*i.e.,* position and velocity) of that joint. In the following, we will refer to the *world frame* as the inertial coordinate frame attached to the ground and the *base frame* as the frame attached to the floating base. For legged robots, the base frame is conventionally oriented with the x-axis pointing forward, y-axis pointing leftward, and z-axis pointing upward the vehicle, as depicted in Figure 3.2b. In this section, we will describe the two state-of-the-art techniques for proprioceptive state estimation on legged robots: strapdown inertial navigation, leg odometry, and the fusion of the two.

2.1.1 Strapdown Inertial Navigation

Inertial navigation is a technique for tracking position, velocity and orientation of a moving vehicle by means of *accelerometers* and *gyroscopes*. An *accelerometer* is a device that measures proper acceleration $\mathbf{a}$, which differs from coordinate acceleration $\ddot{\mathbf{x}}$ because it includes the effect of gravity. An accelerometer at rest measures a non-zero value due to the acceleration of Earth's gravity, yet it measures a zero value when in free fall. A *gyroscope* is a device which measures angular velocity. Its name stems from the original device used for the measurement, which uses the law of conservation of angular momentum, by means of spinning wheels. Nowadays, gyroscopes use different physical phenomena to measure angular velocity. The most common are Micro Electro Mechanical System (MEMS) gyroscopes, which use the Coriolis effect by means

of vibrating structures, and Fiber Optic Gyroscopes (FOGs), which use the Sagnac effect [110]. The performance (and price) of these two devices can differ drastically, therefore it determines their target application [50].

An Inertial Measurement Unit (IMU) is a device which contains three orthogonal accelerometers and three orthogonal gyroscopes, whose signals can be processed to estimate the state of a vehicle with strapdown inertial navigation techniques. The term *strapdown* indicates that the sensor is rigidly attached to the vehicle's chassis, instead of being mounted on a gimbal. From the proper acceleration ${}_i\mathbf{a}_i \in \mathbb{R}^3$ and angular velocity ${}_i\boldsymbol{\omega}_i \in \mathbb{R}^3$, defined in and referred to the IMU frame i, we can compute the corresponding quantities ${}_b\mathbf{a}_b$, ${}_b\boldsymbol{\omega}_b$, referred to the base frame of the vehicle [32]:

$$ {}_b\boldsymbol{\omega}_b = R^b_i \, {}_i\boldsymbol{\omega}_b = R^b_i \, {}_i\boldsymbol{\omega}_i \tag{2.1} $$

$$ {}_b\mathbf{a}_b = R^b_i \, {}_i\mathbf{a}_i \underbrace{- (R^b_i \, {}_i\dot{\boldsymbol{\omega}}_i) \times \mathbf{t}^b_i}_{\text{angular}} \underbrace{- (R^b_i \, {}_i\boldsymbol{\omega}_i) \times \left[(R^b_i \, {}_i\boldsymbol{\omega}_i) \times \mathbf{t}^b_i \right]}_{\text{centripetal}} \tag{2.2} $$

where R^i_b, $\mathbf{t}^b_i$ are the rotation matrix and the rigid translation from the IMU frame i to the base frame b, respectively. To simplify the computation, typically the angular and centripetal components of the acceleration are either neglected [83] or eliminated by choosing the base frame to be coincident with the IMU frame [17]. From ${}_b\mathbf{a}_b$ and ${}_b\boldsymbol{\omega}_b$ we proceed with the estimation of orientation, velocity, and position, by "integration" of these signals [147].

Attitude Estimation

Let us consider the absolute robot orientation ${}_w\Theta_b(t) = \Theta(t) \in \mathrm{SO}(3)$ at time t. Then, for the small angle approximation, the corresponding rate $\dot{\Theta}(t)$ can be expressed in terms of $\Theta(t)$, as:

$$ \dot{\Theta}(t) = \Theta(t) \boldsymbol{\omega}^{\wedge}(t) \tag{2.3} $$

where:

$$ \boldsymbol{\omega}^{\wedge}(t) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & \omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{2.4} $$

is the skew-symmetric matrix of the base angular velocity $\boldsymbol{\omega}(t) = {}_b\boldsymbol{\omega}_b(t)$ at time t. The solution of the differential Equation 2.3, computed at time $t + \Delta t$, is the following [147]:

$$ \Theta(t + \Delta t) = \Theta(t) \exp\left(\int_t^{t+\Delta t} \boldsymbol{\omega}^{\wedge}(t) \mathrm{d}t \right) \tag{2.5} $$

Since the device is providing a discrete signal, we pass from the continuous time variable $t \in \mathbb{R}$ to discrete timesteps $k \in \mathbb{N}$. The integral of Equation 2.5 is then computed using the rectangular rule:

$$ \Theta_k = \Theta_{k-1} \exp(\boldsymbol{\omega}^{\wedge}_k \Delta t) \tag{2.6} $$

where the time interval $\Delta t = t_k - t_{k-1}$ corresponds to the time elapsed between two timesteps. To finalize the transition function from timestep $k-1$ to k, we compute the matrix exponential using Rodrigues' rotation formula:

$$\Theta_k = \Theta_{k-1}\left(I_3 + \frac{\sin\sigma}{\sigma}\boldsymbol{\omega}_k^{\wedge} + \frac{1-\cos\sigma}{\sigma^2}(\boldsymbol{\omega}_k^{\wedge})^2\right) \tag{2.7}$$

where $\sigma = |\boldsymbol{\omega}_k \Delta t|$, and Δt is the time elapsed between two samples at timesteps $k-1$ and k.

Equation 2.7 shows how to estimate the attitude in a rotation matrix form. For estimation of quaternion representation and a more detailed analysis on the attitude estimation algorithm, see [124].

Velocity and Position Estimation

Given the current proper acceleration $\mathbf{a}_k = {}_b\mathbf{a}_{b,k}$ and the current orientation Θ_k (at timestep k) from Equation 2.7, the absolute robot acceleration $\ddot{\mathbf{x}}_k = {}_w\ddot{\mathbf{x}}_{b,k}$, velocity $\dot{\mathbf{x}}_k = {}_w\dot{\mathbf{x}}_{b,k}$, and position $\mathbf{x}_k = {}_w\mathbf{x}_{b,k}$ can be computed again with the rectangular rule:

$$\ddot{\mathbf{x}}_k = \Theta_k \mathbf{a}_k - {}_w\mathbf{g} \tag{2.8}$$

$$\dot{\mathbf{x}}_k = \dot{\mathbf{x}}_{k-1} + \Delta t \ddot{\mathbf{x}}_k \tag{2.9}$$

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \Delta t \dot{\mathbf{x}}_k \tag{2.10}$$

Note that in Equation 2.8 we subtract the gravitational vector ${}_w\mathbf{g} = [0, 0, 9.80655]^{\mathrm{T}}$ to convert from proper to absolute acceleration.

Error Propagation

Equations 2.7, 2.9, and 2.10, provide a full state estimate of the robot's base. However, gyroscopes and accelerometers are affected by multiple types of error, which are quickly propagated to the integrated states. For gyroscopes, the error types include: constant bias, scale factor error due to aging, nonlinearity, scale-factor sign asymmetry due to amplifier mismatch, deadzone due to stiction or fiber lock-in, and quantization error [50]. For accelerometers, we can have: bias, white noise, temperature effects, calibration errors, and bias instability [147]. Note that gyro errors affect not only attitude estimation, but also position and velocity (which depend on it). An attitude error causes an incorrect projection of the gravity vector on the acceleration vector. The acceleration vector is subsequently integrated, resulting in an incorrect estimate of velocity and position as well.

To maintain a balance between the simplicity of the calculation and the modeling accuracy, in practice only the dominant sources of error are modeled, which include: the constant biases, random walk error on the bias terms, and random walk error on the measurements (raw acceleration and angular velocity).

Strapdown inertial techniques alone are inadequate for state estimation of legged robots, but they are successfully used in combination with other sources, as shown in Section 2.1.3.

2.1.2 Early Proprioceptive Sensor Fusion

Leg Odometry (LO) can be defined as the process of estimating the incremental motion of a legged robot, by means of forward kinematics applied to the feet in stable contact with the ground. The term was created in analogy with the traditional odometry for wheeled vehicles. However, in contrast with wheeled vehicles, legged robots have intermittent contact with the ground, and only the legs in contact with the ground are actually propelling the robot. This requires a way to detect the stance legs (*i.e.,* legs in non-slipping contact with the ground) when performing the estimate.

The pioneering work on LO was presented by Roston and Krotkov [118, 117] for Ambler, a massive hexapod robot (mass of $2500\,\mathrm{kg}$ and height of $5\,\mathrm{m}$), designed for space exploration (Figure 2.1a). Their approach consisted of calculating the position of the feet in the base frame, detecting slipping or swinging legs, and using the other legs for estimating the robot's pose. The pose was calculated as the rigid transformation which minimizes the square error between the current feet poses in the base frame, and the estimated feet positions in the world frame. Since the robot was performing only statically stable motions, with one leg moving at a time, the problem was overconstrained. Thus, the solution to the minimization problem was found by Singular Value Decomposition (SVD).

Several years later, Lin *et al.* [105] applied a similar approach to a much smaller ($50\,\mathrm{cm}$ long, $7\,\mathrm{kg}$ mass) hexapod robot, called RHex [122] (see Figure 2.1b). This time, the gait was designed to move with three legs at a time, but with a sufficiently long double support phase (*i.e.,* with all legs on the ground) in between one tripod configuration and the other. Since the robot was moving only when all the legs were on the ground, the pose was estimated by composition of the transformations collected at the beginning and at the end of each double support phase. Due to the absence of contact sensors, to detect the double support phase, the authors designed an algorithm which detects when all the feet lie on the same plane. Therefore, this leg odometer implicitly assumes that the terrain is locally planar. Later, the authors extended their previous work to dynamic gaits [106, 80], and presented one of the first attempts of sensor fusion on state estimation of legged robots. The fusion approach was motivated by the fact that the jogging gait executed on RHex has a flying phase, when no legs are in contact with the ground. The gait is decomposed into four different phases: tripod stance, liftoff transient, aerial, and touchdown transient. To properly handle the aerial and transient phase, the authors proposed the use of a 12-axis accelerometer suite, made of four 3-axis accelerometers located at different places on the robot. This allowed them to compute the base angular acceleration (see Equation 2.2), useful to compute second order dynamics.

At the same time, another work on an hexapod, Lauron III [46], was presented by Gaßmann *et al.* [47]. The robot was equipped with a Global Positioning System (GPS) receiver, an IMU,

joint encoders and foot sensors. Joint loads were inferred from motor currents. In this work, the authors used the fuzzy logic theory to extend the concept of contacts from a binary state to a continuous interval $[0, 1]$. The fuzzy weights were computed taking into consideration three leg states: ground contact, slippage and collision. Each state was inferred from the sensors through a simple heuristic (*e.g.,* when the absolute force at the end effector is high, the ground contact is high), and the three states were combined, again heuristically, to determine the fuzzy weight. The work was one of the first fusing LO outputs with other sources. In this case, the pose and orientation from the leg odometer were fused with GPS signals on an error state Kalman Filter (KF).

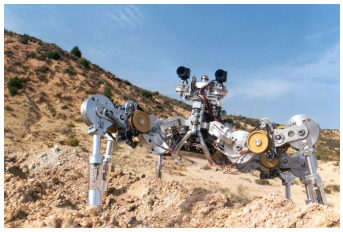
A few years later, Chitta *et al.* [31] proposed a sensor fusion solution based on the Particle Filter (see Appendix B.4) for the localization of the small quadruped LittleDog (see Figure 2.1d) on known terrain. The robot executed a static planned motion composed of four stages: leg selection (depending on the sequence), foothold prediction, foothold execution, and body motion. Given the high computational expense of particle filters at high dimensionality, the state of the particle filter was reduced from six to three DoFs: forward displacement, lateral displacement, and yaw. Since the gait is static, the reduced state was propagated forward when the robot was in quadruple support, as an incremental planar roto-translation. Since the gait was static, velocity and acceleration were not part of the state. Roll and pitch were computed directly from the IMU measurements. The particle filter and the IMU imposed constraints on five DoFs out of six. The remaining DoF, *i.e.,* the height off the ground, was computed using the knowledge of the terrain and the constraints of the leg configurations. Since the terrain was known, the height of each particle was set such that the robot would not penetrate the ground. Then, when the robot was in triangular support, the swing leg was moved until it was in contact with the ground. The error between the leg configuration at the predicted height and the actual leg configuration in the four support state was used as a measurement update for the particle filter.

Reinstein *et al.* [114, 115] continued the work on proprioceptive sensor fusion by proposing an Extended Kalman Filter (EKF) approach with a new type of leg odometer. The method have been applied to Puppy, a small quadruped robot with four actuated joints at the hips and four passive joints at the knees. The robot was equipped with a MEMS IMU, joint encoders, and pressure sensors at the feet. The absence of hip joints for adductive/abductive motions imposed lateral non-holonomic constraints which were exploited for the odometry computation. The proposed LO method computed only the stride length by regression of a set of features, including: amplitude, mean, variance and numerical integral of the raw signals. The feature set also included linear and nonlinear combinations of the above features, for a total of approximately 200 features. The training phase was executed with standard batch gradient descent on the quadratic error function. The estimated stride length was used as a measurement update of the EKF, while the prediction step of the filter was executed by a strapdown inertial navigation algorithm [124, 125] (see Section 2.1.1).

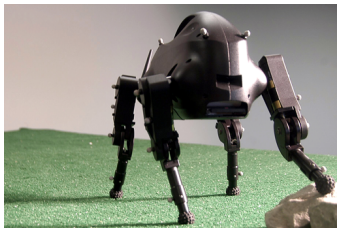
Görner and Stelzer [48] have shown a proprioceptive leg estimator for their small hexapod, the DLR crawler [49] (Figure 2.1e). In contrast with other works, the robot's pose was estimated

(a) Ambler

(b) RHex



(c) Lauron III



(d) LittleDog

(e) DLR Crawler

Figure 2.1: Robots used in the early stages of leg odometry.
(a) Ambler [78] (Carnegie Mellon University, http://www.ri.cmu.edu);
(b) RHex [122] (Boston Dynamics, http://bostondynamics.com);
(c) Lauron III [46] (Forschungszentrum Informatik Karlsruhe, http://www.fzi.de);
(d) LittleDog (Boston Dynamics, http://bostondynamics.com);
(e) DLR Crawler [49] (DLR, http://www.dlr.de)

by using both joint angles and measured torques, without using an IMU. The proposed method estimated the pose by matching the contact points of the feet on the ground (considered as "point clouds") at two consecutive time steps. Therefore, the algorithm required at least three non-collinear feet in contact with the ground to work. It also assumed a static gait. In this way, the only force acting on the robot (excluding external ones) was gravitational, so as to estimate roll and pitch by reconstruction of the gravity vector from the joint torques.

All the works presented above show a significant progress towards a general solution for proprioceptive estimation on legged machines. However, none of these have shown sufficient generality in terms of gait, number of feet on the ground, and type of terrain. In the next section, we explore a variety of general proprioceptive approaches to legged state estimation.

2.1.3 Kinematics-Inertial State Estimation

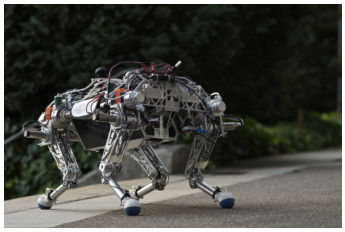
A general solution for proprioceptive sensor fusion on legged machines was proposed by Blösch *et al.* [17] on the medium sized quadruped robot StarlETH [65] (Figure 2.2a). In their approach, they proposed an EKF state estimator that included an inertial process model for the Kalman prediction phase and forward leg kinematics as measurement update. The approach is general in the sense that it is not restricted to specific gaits, terrain types, or leg configurations. The base state includes position, velocity, orientation, acceleration bias, angular velocity bias, and absolute feet contact points. The inclusion of the feet positions expressed in the base frame is a key feature of the approach. In a certain way, when a foot touches the ground, the robot is "scanning" the terrain with its own leg. The contact states of the feet are detected through contact sensors: whenever a contact sensor detects the swinging phase of a leg, the covariances of that leg are set to infinity, and the filter ignores the leg for the computation. For the remaining legs $l \in C$, which are in contact with the ground, the forward kinematics provides the corresponding feet coordinates ${}_b\mathbf{x}_{f_l}$ expressed in the base frame. Since the foot is fixed to the ground at that moment, this measure is equivalent to the difference between the absolute foot contact position ${}_w\mathbf{x}_{f_l}$ and the absolute base position ${}_w\mathbf{x}_b$:

$$ {}_b\mathbf{x}_{f_l} = \Theta^b_w({}_w\mathbf{x}_{f_l} - {}_w\mathbf{x}_b) \tag{2.11} $$

Equation 2.11 provides a measurement for the Kalman update process. The authors also provide a nonlinear observability analysis, which confirms the non-observability of the linear position and yaw states, and highlights the degenerate cases when rank loss occurs (*e.g.,* when three feet are collinear and $\boldsymbol{\omega} = \ddot{\mathbf{x}} = 0$). For non-degenerate cases, only one foot in contact is sufficient to reach the observability region, if the angular velocity axis is not perpendicular to gravity and the acceleration is non-zero. The validity of this approach was demonstrated on the medium-sized quadruped robot StarlETH (see Figure 2.2a) for a $1\,\mathrm{min}$ straight crawl and a simulated trot.

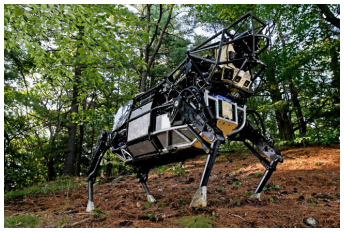
Rotella *et al.* [119] extended the work of [17] to a humanoid robot with flat feet. In this work, the state vector is augmented to include the orientation of the flat foot, which is used to constrain the other states. The validity of the approach has been shown for a $2\,\mathrm{min}$ long simulated walk. Lubbe *et al.* [81] also extended the application of [17] by implementing the developed methodology on a commercial hexapod robot, equipped with industrial grade IMU and contact sensors. The approach was tested on a quasi-static tripod gait over flat terrain. The system was controlled to follow a squared path (edges of $1\,\mathrm{m}$) without active body turning. They reported a position drift of approximately $5\,\%$ of the distance traveled, which validated the use of the framework on commercial hexapods.

In [16], Blösch *et al.* extended their previous work by replacing the EKF with an Unscented Kalman Filter (UKF) (see Appendix B.3), and redefining the formulation of the state vector to be robot-centric (*i.e.,* with quantities referred to the base frame). The absolute position of the contact points is no longer part of the filter: the measurement update is now computed from the

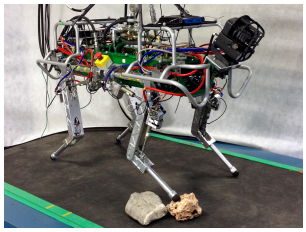


(a) StarlETH

(b) BigDog

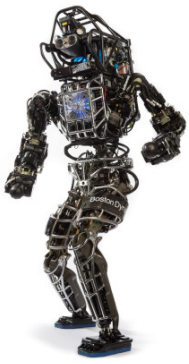


(c) LS3



(d) HyQ

Figure 2.2: Examples of quadruped robots.
(a) StarlETH (ETH Zürich, http://www.rsl.ethz.ch);
(b) BigDog (Boston Dynamics, http://bostondynamics.com);
(c) LS3 (Boston Dynamics);
(d) HyQ (IIT, http://iit.it/hyq), the experimental platform used in this thesis.



(a) Atlas

(b) SARCOS Humanoid

(c) Valkyrie

Figure 2.3: Examples of humanoid robots.
(a) Atlas (Boston Dynamics);
(b) SARCOS Humanoid (SARCOS, http://www.sarcos.com, http://www-clmc.usc.edu);
(c) Valkyrie (NASA, http://nasa.gov).

zero-velocity constraint of the leg in contact with the ground (*cf.* with Equations 4.9 and 4.10):

$$\underbrace{-{}_b\dot{\mathbf{x}}_b}_{\text{base velocity}} + \underbrace{{}_b\boldsymbol{\omega}_b \times {}_b\mathbf{x}_{f_l}}_{\text{rotational velocity}} + \underbrace{J_l(\mathbf{q})\dot{\mathbf{q}}_l}_{\text{foot velocity (rel. to base)}} = 0 \tag{2.12}$$

The choice of the UKF was motivated by the dependency on the angular velocity of both the inertial prediction and the measurement update Equation 2.12. Thus, there is a correlated noise between the prediction and the measurement update phase of the filter. With the UKF setup, this situation is handled by proper choice of the sigma points of the filter. To remove measurement outliers, the velocity contributions are discarded if the filter innovation, expressed in Mahalanobis distance, exceeds a fixed threshold, which is found empirically. This procedure eliminates the effect of spurious filter updates originating from the legs whose contact state estimates are unreliable. As in [17], the performance of the approach was demonstrated on the StarlETH platform, for a 23 s trot on a flat terrain made unstable and slippery by placing wooden debris along the way.

An alternative approach to the inertial process model is given by methods based on dynamic models. Stephens [138] showed that a Center Of Mass (COM) state estimator based on a 1D Linear Inverted Pendulum Model (LIPM) can improve the estimation of position and velocity of the COM under external forces on the SARCOS Humanoid robot (Figure 2.3b) executing periodic motions.

A more theoretical work on the subject was provided by Gür *et al.* [52]. In their paper, the authors incorporated a Spring Loaded Inverted Pendulum (SLIP) dynamic motion model into the state estimate. They demonstrated, with a simulation on a monopod, how the state can be better tracked when the assumption of constant acceleration between two IMU measurements is replaced with their dynamic model.

Xinjilefu *et al.* [148] proposed to use the full body state dynamics on the Atlas robot (Figure 2.3a). However, given the high number of DoFs, using it in a nonlinear KF framework was computationally unfeasible. Therefore, they separated the estimation of the base and the estimation of the other joint states into two independent filters. The base filter was similar to other approaches, with an inertial process model for the prediction and forward kinematics with zero velocity stance feet constraint for the update. This work was later extended by the same authors in [149]. Instead of using an EKF, they proposed to formulate the state estimation problem as a Quadratic Programming (QP) problem. In this way, the full model dynamics could be incorporated into the estimation, without expressing it into the state space, as a Kalman filter would require. The estimation was reformulated as an optimization problem which minimizes a quadratic form of the modeling error and the measurement error, weighted according to the uncertainties of the model and the measurements. Tests on the Atlas robot have shown that the chattering due to corrective actions from the feedback controller were reduced by half.

2.2 Contact Estimation Methods

Most research in the area of contact estimation is focused on collision avoidance for safe Human Robot Interaction (HRI) with manipulators. De Luca *et al.* [82] proposed a collision detection and reaction method which identifies external forces acting on a link as first order filtered external torques acting on the manipulator's joints. The reaction strategy typically involves stopping or moving the link away and along the direction of the identified contact. Haddadin *et al.* [53] extended this work by introducing a modified version of the contact detection, introducing more recovery strategies, and extensively experimenting with a human subject.

More recently, Hwangbo *et al.* [67] developed a probabilistic contact estimator to control the quadruped electric robot ANYmal [66] without foot sensors. The method fuses information about dynamics, differential kinematics and kinematics in a way similar to a Hidden Markov Model (HMM) to reconstruct the contact status. The validity of this approach was demonstrated by comparing their method with Generalized Momentum (GM) approaches, using the delay detected by OptoForce sensors as performance metric.

It has to be noted that the contact detection methods presented above aim to detect the contact as early as possible, in order to promptly take countermeasures against unwanted collisions [82, 53] or to control the robot [67]. In contrast, we are interested in detecting the first instant of a leg's contact phase from which a reliable and trustworthy velocity measure can be produced, which is a substantially different goal. In Chapter 4 we describe how to achieve this by: 1) learning the threshold of the normal component of the GRF that minimizes the velocity error; and 2) incorporating impact information and consistency between feet velocity estimates in the computation of the covariance of the filter measurement update.

2.3 Exteroceptive Pose Estimation Methods

In the previous section, we described the state-of-the-art methods for proprioceptive state estimation of legged robots, starting from the early works in LO, to the fusion with joint kinematics, dynamics and inertial information. In this section, we introduce exteroceptive inputs in the estimation process. We first describe the exteroceptive pose estimation techniques alone and then we review the current methods for accurate state estimation fusing proprioceptive and exteroceptive sources.

2.3.1 Visual Odometry

Visual Odometry (VO) is the process of estimating the egomotion of a robot using the input of one or more cameras attached to it. The concept dates back to 1980, with the seminal work of Moravek [93]. Since then, for more than two decades, most of the research efforts on the topic were made by NASA and JPL, in preparation of the upcoming Mars Exploration Rover mission. These works culminated with the successful use of VO on another planet. The algorithm was

used for multiple years by the Spirit and Opportunity rovers, to track their pose and compensate for wheel slippage on the sandy surface of Mars [87].

The term *visual odometry* actually appeared for the first time only in 2004, with the work of Nister [100]. It was chosen in analogy with wheel odometry. Wheel odometry incrementally estimates the motion of a vehicle by integrating the number of turns of its wheels over time. Similarly, VO incrementally estimates the pose of a vehicle by tracking the changes that motion induces on the images of its onboard cameras [127].

VO works under the assumption that the scene captured is sufficiently illuminated, static, texturized, and overlapping with previously captured ones (*i.e.,* small incremental motion between consecutive frames). These ideal conditions are rarely met in real scenarios, where image blur due to vibrations and sharp motions, scarce illumination, and dynamic scenes are pervasive (see for example, Appendix A.3). For these reasons, the proprioceptive estimation techniques introduced in the previous sections can provide a valid support when VO fails [83].

The working principle behind VO is based on tracking the relative motion of the objects in a scene from one camera frame to the next. If the scene captured is static, this is the equivalent of tracking the camera motion within the environment. When concatenated, the consecutive frame-to-frame transformations provide an estimate of the full trajectory of the camera (*i.e.,* the robot's pose, since the camera is rigidly attached to it). The VO methods can be divided into:

Direct methods: the transformation is given by the solution which minimizes the per-pixel intensity difference between the two images. Direct methods are further subdivided in three categories [69]: dense (full picture), semi-dense, or sparse;

Feature based methods: the algorithm detects some feature points (*e.g.,* corners) in one image, and searches for their corresponding ones in the following image. The relative motion between the two images is given by the solution which minimizes the reprojection error of one feature set onto the other. If the features are detected in the first image and then tracked in the following ones, with local search techniques, the method is said to be *feature tracking* based. Conversely, if the features are independently detected in the two images, and then matched based on similarity metrics of their descriptors, the method is said to be *feature matching* based.

A further classification of VO is based on the number of cameras used: most of the research on VO has been performed on stereo cameras. This allows to operate in the 3D domain directly, whereas in monocular VO the problem is underconstrained since the scale is unknown and multiple views are required to estimate it. For a complete overview on VO, we invite the reader to consult [127, 44], and the recently released (year 2016) survey by Cadena *et al.* [24].

In legged robotics, most of the VO methods adopted are feature-matching based and use stereo cameras. In 2002, Hirschmüller *et al.* [58] proposed a novel stereo VO which relied on feature matching instead of feature tracking. The method leverages the rigidity constraint of 3D motion to discard most of the outliers, and limits the effect of remaining ones. The method was later used on the DLR Crawler (Figure 2.1e), as described in Section 2.4.

Inspired by [58], Howard [62] presented a VO for two mobile platforms, one of which is the dynamic legged robot BigDog [107] (Figure 2.2b). His approach is based on feature detection and matching on the points which have a valid stereo disparity value, using standard corner detectors. Each detected feature point is then associated to a descriptor given by its neighborhood. The feature matching is performed by evaluating the Sum of Absolute Distance (SAD) score between all the feature pairs. Since all the pairs are composed by 3D points (from 2D coordinates and disparity), the matching feature points can be further filtered by taking into account rigidity constraints of the motion. The final set of inliers is computed as the largest set of correspondences which respect the rigidity constraints. Finally, the motion is computed by minimization of the reprojection error using the Levenberg-Marquardt least squares algorithm. The method was not tested on the real robot, but rather on the sensor head, manually transported on a sandy testbed.

An almost identical approach to [62] was shown by Huang *et al.* [64] for autonomous flight of a quadrotor equipped with a Kinect sensor (see Section 5.5.1). This method was later fused with other sensor sources on the Atlas robot [37], in preparation for the DARPA Robotics Challenge (DRC), but not used in the actual competition.

2.3.2 Visual SLAM

The Simultaneous Localization and Mapping (SLAM) problem for a mobile robot is defined as building a consistent map of the environment while simultaneously determining the robot's location within this map [33, 6].

As the name suggests, Visual SLAM (V-SLAM) aims at solving the SLAM problem using visual inputs (monocular cameras, stereo cameras, RGB-D sensors). The main difference between VO and V-SLAM is that the former typically operates on a frame-to-frame basis and is prone to drift, while the latter keeps a history of the past poses and visual features. These are used for graph-based trajectory optimization [71, 73] and detection of loop closures (*i.e.,* situations when the robot returns to a past location) in order to eliminate the drift accumulated over long periods of time. These techniques allow to reconstruct the full trajectory of the camera and the scene it captured.

Some examples of V-SLAM algorithms include ORB-SLAM [94, 95] and LSD-SLAM [35]. To work properly, V-SLAM methods should be applied under the same (or better) conditions where VO is normally used. For this reason, V-SLAM is not common in legged robotics and its direct application to legged robotic navigation is not straightforward.

Since we are more interested in local mapping, in this thesis we will not cover explicitly V-SLAM for legged robots, even though it is subject to future work (see Chapter 7).

2.3.3 Iterative Closest Point

The Iterative Closest Point (ICP) [13] is a well known algorithm for estimating the rigid transformation that aligns two point clouds capturing the scene, but from different points of view.

This alignment process is known as *point cloud registration.* Given its simplicity and broad applicability, the ICP was quickly specialized into a variety of applications, including: 3D object reconstruction, non-contact inspection, medical and surgery support, and autonomous vehicle navigation. We are interested in the last of these applications. In this section we provide a brief description of the algorithm, and a literature review on the application to mobile legged platforms. A complete survey on the method and its multiple derivations was provided by Pomerleau *et al.* in [109]. A review on the mobile robotic applications of ICP is also available in [108], by the same authors.

Algorithm Description

The algorithm defines:

- a *target* point cloud $\mathcal{C}_t^{[\mathrm{t}]} \in \mathbb{R}^{N\times 3}$, whose N points are expressed in the target frame of reference t, and are kept unaltered;
- a *source* point cloud $\mathcal{C}_s^{[s]} \in \mathbb{R}^{N\times 3}$, whose N points are transformed from the source frame of reference s to the target frame of reference t, through the rigid transformation ${}_t\mathrm{T}_s$

The two point clouds can then be merged together into a third one by summing the points of the target with the transformed points of the source:

$$\mathcal{C}_t^{[\mathrm{m}]} = \mathcal{C}_t^{[\mathrm{t}]} + {}_t\mathrm{T}_s\mathcal{C}_s^{[\mathrm{s}]} = \mathcal{C}_t^{[\mathrm{t}]} + \mathcal{C}_t^{[\mathrm{s}]} \tag{2.13}$$

The goal of ICP is to estimate the transformation ${}_t\mathrm{T}_s$ through an iterative process. We can summarize it as follows:

1. For each point of $\mathcal{C}_s^{[\mathrm{s}]}$, find the closest point from the cloud $\mathcal{C}_t^{[\mathrm{t}]}$;
2. Compute the ${}_t\mathrm{T}_s$ which minimizes the mean squared distance between the previously defined point pairs;
3. Transform the points of ${}_t\mathrm{T}_s$ with the estimated transformation;
4. Repeat from 1 until convergence.

Since the two point clouds are captured from different locations, the samples from the source cloud do not have a perfect correspondence in the target cloud. For this reason, Chen and Medioni [29] proposed to replace the point-to-point distance metric with the point-to-plane metric. This was one of the first examples of 3D feature extraction to improve the algorithm in structured environments. Further studies in this direction include: the Generalized-ICP, by Segal *et al.* [129], which combines the standard ICP and its point-to-plane variant into a single probabilistic framework; and the Normal ICP, by Serafin and Grisetti [133], which incorporates both normal and curvature information into the point association.

Inertial-Registration Fusion

The ICP algorithm is known to be sensitive to initial conditions: if the two clouds are too far from each other, the algorithm will likely diverge or stop on a local minimum. Given the wide diffusion of low-cost MEMS IMUs, several methods which take advantage of inertial measurements have been proposed.

Nießner *et al.* [98] combined inertial measurements and depth sensor outputs for dense scene reconstruction. Their framework includes a method for detecting an ICP failure and switching to dead reckoning by IMU integration, to avoid bad reconstructions.

Qayyum and Kim [112] fused inertial and depth sensor information with a modular EKF framework. They addressed the practical development of an Inertial-Kinect fused SLAM that works outdoor. They focused on handling the 3D to 2D degeneration in structured light sensing, called the depth dropout problem.

Bethencourt and Jaulin [14] recently introduced a new concept for point cloud registration based on an interval analysis method. They are not focusing on SLAM, but achieved consistent Kinect point cloud matching using the IMU data.

2.3.4 Normal Distribution Transform

In 2003, Biber and Straßer [15] introduced an alternative approach to ICP for the scan matching of 2D laser scans. The authors called their method Normal Distribution Transform (NDT). The space is regularly divided into cells of equal size. For each cell containing at least three points, the mean and covariance matrix of the points is computed. The probability of measuring a new sample on a cell is thus modeled by the normal distribution parametrized by the mean and covariance of that cell. Given two scans, the algorithm computes the relative motion between the two by optimizing the sum of the normal distributions for all the cells using the Newton's algorithm.

The NDT was later extended to the 3D domain by Magnusson [85], who applied the method to the autonomous navigation of mining vehicles. To overcome the sparsity due to the increased dimensionality, the regular grid partitioning is replaced by iterative subdivision techniques [86].

2.4 Multisensor State Estimation

Chilian *et al.* [30] proposed a multi-sensor fusion algorithm which uses the related works in leg proprioceptive estimation [48] and VO [58] for the DLS Crawler. The LO and VO modules, together with inertial information, are fused into an indirect feedback Information Filter (IF). The choice of this filter, which is the dual version of the KF, was motivated by the author because it can easily handle concurrent inputs. The state maintained by the filter is an error state, which contains, as usual: position error, velocity error, orientation error, and biases' error. The fused state showed a good accuracy (1.1% error of distance traveled) for the periodic motion inside a test bed filled with gravel.

A very similar approach was adopted by Ma *et al.* [84, 83], which fused the information from a stereo camera, leg kinematics, and a tactical grade IMU, using an error state EKF instead of an IF. Following common inertial navigation techniques from avionics, the state vector is defined as a vector of position, velocity, attitude and IMU biases. The approach is focused on visual inertial fusion with LO measurements expressed as delta positions between two key frames, used only in case of failure of the Visual Odometry (VO). The approach produced a robust performance with an error below $1\,\%$ of the distance traveled when fused with GPS.

Fallon *et al.* [37] presented Pronto: an efficient, modular and open-source EKF-based state estimator. The algorithm uses an IMU-based process model and combines this with measurement corrections from different sensor modalities (LO, LiDAR and VO) to produce a position estimate for the humanoid robot Atlas, developed by Boston Dynamics. The inertial process model and the Gaussian Particle Filter (GPF)-based LiDAR modules are the same as presented in [22]. The LO module handles contacts using a Schmitt trigger (a two threshold comparator with hysteresis, see [104]) on the contact sensor signals: the contact is detected when the low threshold is crossed and it is released when an arbitrary time has passed and the high threshold is crossed. When the robot is in double support, only one leg is used for simplicity. The filter is able to handle out-of-order and asynchronous inputs from different sensors. Originally developed for the MIT DARPA Robotics Challenge team, its implementation has been recently released publicly[2].

Recently, Nobili *et al.* [101] proposed the Auto-tuned Iterative Closest Point (AICP), a robust pose estimation algorithm for the humanoid robots Valkyrie (Figure 2.3c) and Atlas (Figure 2.3a). For both robots, the spinning LiDAR from the Multisense SL tri-modal sensor (Figure 3.4e) was used as input. The LiDAR scans are accumulated using inertial and kinematics information fused with the EKF of [37]. The method has a pre-filtering phase, where only the planar portions of the accumulated LiDAR scans are retained. In this way, only strong 3D features are kept, the dynamic parts of the scene (*e.g.,* the people) are eliminated.

Although focused mostly on the controller, the work by Koolen *et al.* [76] dedicates a section to state estimation on the Atlas robot for the DRC. The authors claim that, when combined with an ICP registration algorithm, their state estimation algorithm from [75] can reduce the drift from $2\,\mathrm{cm}$ per step to almost zero. However, the ICP based LiDAR registration was not used during the competition.

2.5 Mapping for Legged Robot Navigation

In this section, we describe a number of mapping methods (online and real-time) for legged robot navigation. These include elevation maps, meshes, OctoMaps, and point clouds.

When choosing an appropriate mapping representation for a mobile robot, one has to consider three aspects [136]:

1. The precision of the map must match the precision required by the robot to achieve its

[2]https://github.com/ipab-slmc/pronto-distro

goals;

2. The precision of the map must match the precision of the data returned by the robot's sensors;

3. The complexity of the map representation has direct impact on the computational complexity of the algorithms which process it.

In other words, a map suitable for navigation should contain enough detail to execute the task at hand (*e.g.,* footstep planning), yet should be simple enough to be processed by the navigation algorithms.

2.5.1 Elevation Map

The majority of mapping approaches for legged robots on rough terrains involve the use of elevation maps for body path planning or tele-operation. This approach has become popular for wheeled vehicles, especially planetary rovers [88], and has also been adopted for legged robots.

In elevation maps, the environment is represented as a discretization of a function $h = f(x, y)$, where x and y are the planar coordinates of a reference plane (typically gravity aligned) and h is the corresponding elevation of the terrain from that plane [23]. The xy plane is usually discretized in a regular grid of cells, which can be efficiently stored in a 2-D array. Each value of the array contains a summary of the measurements collected within the area of the corresponding cell.

Stelzer *et al.* [137] developed a complete visual navigation framework for their hexapod robot. The algorithm used stereo images from which depth images were computed. Pose estimates are obtained by fusing inertial data with relative leg odometry and visual odometry measurements using an indirect information filter.

With quadruped robots, a few groups have also presented some localization/state estimation solutions. For example, Kolter *et al.* [74] used a stereo camera with a simple ICP-based technique for offboard point cloud registration to incrementally build a map. Then, they used a texture synthesis algorithm to fill occluded areas in order to perform motion planning with LittleDog.

Bajracharya *et al.* [7] presented a stereo mapping approach which uses a hybrid 2D/3D data structure. Developed under the DARPA LS3 program, this mapping algorithm attains a map fidelity of $5\,\mathrm{cm}$ out to $5\,\mathrm{m}$.

Bradley *et al.* [21] used a similar representation for the terrain when using the LS3 robot (Figure 2.2c), but also included 3D blobs to model vertical obstacles, such as trees and bushes, as an additional element to the elevation map.

Fankhauser *et al.* [38] proposed an elevation mapping method which incorporates uncertainties from the sensor and the state estimator [17], by means of a Kalman filter, from a robot-centric perspective. The mapping is obtained by fusing height maps from a Kinect.

2.5.2 Point Cloud

Fallon *et al.* [36] presented a perception and planning algorithm which allowed the Atlas humanoid to continuously step up and down on a pile of cinder blocks with the Atlas humanoid, without stopping. The map used to generate the footsteps was collected online with the Kintinuous algorithm [143] using the point clouds generated from the Multisense SL device. Since the point clouds from stereo are noisier than their depth sensor counterparts, the output from the device is pre-filtered and smoothed. For the actual footstep planning, plane extraction was also applied to the smoothed cloud.

2.5.3 OctoMap

The 3-D equivalent of elevation maps is represented by occupancy voxel maps (*i.e.,* a 3D array of cells). This representation is highly inefficient, due to the intrinsic sparsity of the 3D space. To overcome this limitation, Hornung *et al.* proposed an implementation based on octrees, called OctoMap [61]. This data structure was used as a prior map for the localization in [22, 37, 60] and for foothold planning in [145, 90], as will be detailed in Section 3.8. Although a GPU-accelerated implementation exists [57], it has mostly been evaluated for indoor service robots.

2.6 Summary

In this chapter, we have reviewed several state-of-the art methods to endow a legged robot with motion estimation and mapping capabilities. We have seen that legged machines, especially when employed in difficult operations in GPS-denied environments, can benefit from multi-sensor fusion techniques. The trend towards techniques which use a combination of proprioceptive and exteroceptive state estimation is highlighted in Table 2.1. In Chapters 4 and 5, we describe how the state estimator we developed on Hydraulic Quadruped (HyQ) corresponds with this trend.

For mapping, by far, the most used approach is towards elevation map representations. This appears to be a reasonable tradeoff between complexity and detail: the terrain is always "dense", while occasional overhanging obstacles can be handled by enriching the representation instead of using an explicit voxel representation. Chapter 6 describes in detail the mapping system implemented on HyQ, which consists of a lightweight Point Cloud representation with on-demand heightmap extraction.

Method	Year	Filter	State	LO	IMU	VO	GPS	LiDAR
Roston and Kotkov [117]	1992		$\mathbf{x}, \boldsymbol{\theta}$	✓				
Gaßman *et al.* [47]	2005	EKF	$\mathbf{x}, \boldsymbol{\theta}$	✓			✓	
Lin *et al.* [80]	2006	EKF	$\mathbf{x}, \boldsymbol{\theta}$	✓	✓			
Chitta *et al.* [31]	2007	PF	$\mathbf{x}, \boldsymbol{\theta}$	✓			✓	
Stephens [138]	2011	KF	$\mathbf{x}, \boldsymbol{\theta}$	✓				
Reinstein *et al.* [114]	2011	EKF	$\mathbf{x}, \boldsymbol{\theta},$	✓	✓			
Chilian *et al.* [30]	2011	IF	$\boldsymbol{\delta}\mathbf{x}, \boldsymbol{\delta\theta}, \boldsymbol{\delta}\dot{\mathbf{x}}, \boldsymbol{\delta}\mathbf{b}_\omega, \boldsymbol{\delta}\mathbf{b}_a$	✓	✓	✓		
Ma *et al.* [84]	2012	EKF	$\boldsymbol{\delta}\mathbf{x}, \boldsymbol{\delta\theta}, \boldsymbol{\delta}\dot{\mathbf{x}}, \boldsymbol{\delta}\mathbf{b}_\omega, \boldsymbol{\delta}\mathbf{b}_a$	✓	✓	✓	✓	
Blösch *et al.* [17]	2012	EKF	$\mathbf{x}, \boldsymbol{\theta}, \dot{\mathbf{x}}, \mathbf{p}, \mathbf{b}_\omega, \mathbf{b}_a$	✓	✓	✓	✓	
Blösch *et al.* [16]	2013	UKF	$\mathbf{x}, \boldsymbol{\theta}, \dot{\mathbf{x}}, \mathbf{b}_\omega, \mathbf{b}_a$	✓	✓	✓	✓	
Rotella *et al.* [119]	2014	EKF	$\mathbf{x}, \boldsymbol{\theta}, \dot{\mathbf{x}}, \mathbf{p}, z, \mathbf{b}_\omega, \mathbf{b}_a$	✓	✓			
Xinjilefu *et al.* [149]	2014	QP	$\mathbf{x}, \boldsymbol{\theta}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$	✓	✓			
Fallon *et al.* [37]	2014	EKF	$\mathbf{x}, \boldsymbol{\theta}, \dot{\mathbf{x}}, \mathbf{b}_\omega, \mathbf{b}_a$	✓	✓	✓		✓
Nobili *et al.* [101]	2017	EKF	$\mathbf{x}, \boldsymbol{\theta}, \dot{\mathbf{x}}, \mathbf{b}_\omega, \mathbf{b}_a$	✓	✓			✓

Table 2.1: Comparison of different state estimation approaches. *Legend:* $\mathbf{x}$ = linear position; $\dot{\mathbf{x}}$ = linear velocity; $\boldsymbol{\theta}$ = orientation, $\boldsymbol{\omega}$ = angular acceleration; $\mathbf{b}_a$ = acceleration bias; $\mathbf{b}_\omega$ = angular acceleration bias; $\boldsymbol{\delta}$ = error state; $\mathbf{q}$ = joint position; $\mathbf{p}$ = foot position; z = foot inclination; $\dot{\mathbf{q}}$ = joint velocity.

Chapter 3

System Overview

HyQ is the robotic platform used for all the experiments presented in this dissertation. In this chapter we describe its main characteristics from a perception-oriented perspective. After a brief overview, most of the attention is dedicated to its sensory system configurations, calibration, and signal synchronization (which are crucial for state estimation and mapping purposes). For a detailed description of the mechanical design, we invite the reader to consult [131, 132].

3.1 Mechanical Design

HyQ is a fully torque-controlled hydraulic quadruped robot. It has been designed to perform a variety of motions, ranging from agile and highly dynamic locomotion to slow, careful motions on challenging terrains [132]. It weighs $85\,\mathrm{kg}$, is $1\,\mathrm{m}$ long and $1\,\mathrm{m}$ tall (see Figure 3.2a). Figure 3.1 shows HyQ's leg and joint configuration: the robot has four legs: Left Front (LF), Right Front (RF), Left Hind (LH), Right Hind (RH). Each leg has three actuated DoFs: two joints, the Hip Flexion-Extension (HFE) and the Knee Flexion-Extension (KFE), rotate around an axis perpendicular to the sagittal plane (red); one joint, the Hip Abduction-Adduction (HAA), rotates around an axis perpendicular to the coronal plane (blue). At the nominal pressure of $16\,\mathrm{MPa}$, these joints can exert a maximum torque ranging from $120\,\mathrm{N\,m}$ for the HAA to $145\,\mathrm{N\,m}$ for the other joints.

3.2 Frames of Reference

Figure 3.2 summarizes the frames of reference adopted throughout this dissertation. The *base frame* is located at the geometric center of the torso, in the plane where all four HAA axes lie. It follows the forward-left-up ground vehicle convention for orientation. The *world frame* (sometimes called *local frame* in literature [84, 83], in contrast to the global coordinates provided by satellite navigation systems) is an inertial frame attached to an arbitrary fixed point on Earth. It is often chosen to be coincident with the base frame when the robot is in its starting position. The *imu frame* is located at the sensor origin of the IMU. Its relative transformation

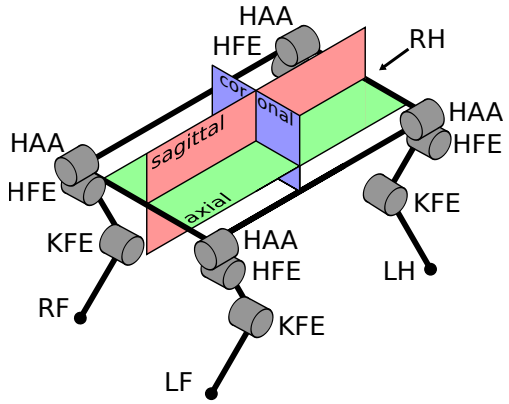


Figure 3.1: Joints names and planes conventions on HyQ. The robot's legs have three actuated degrees of freedom each: two joints rotating perpendicularly to the sagittal plane (red), the Hip Flexion/Extension and the Knee Flexion/Extension; one joint rotates perpendicularly to the coronal plane (blue), the Hip Abduction/Adduction. The legs are identified by the following abbreviations: Left Front (LF), Right Front (RF), Left Hind (LH), Right Hind (RH). The RH foot and its knee joint are occluded.

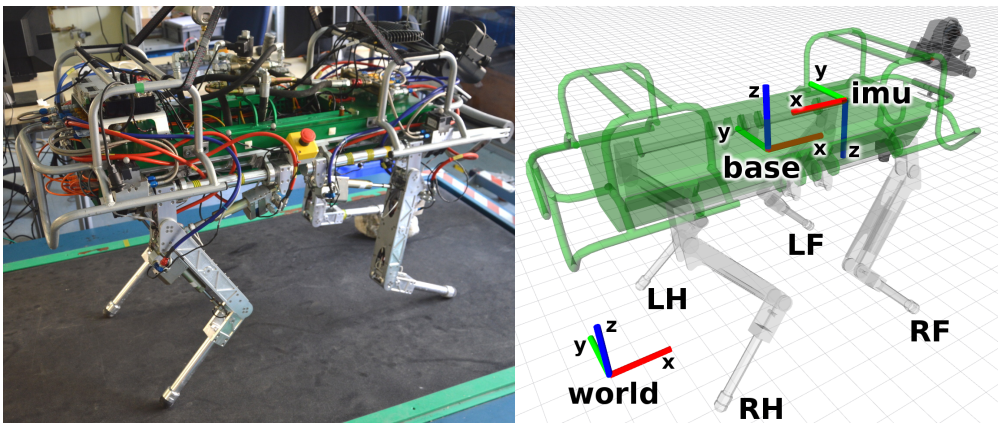
from the base frame is computed based on the CAD model of the robot and sensor datasheet. The *horizontal frame* (not shown in the picture) is an additional convenient frame introduced in [8], for trajectory generation purposes. Its position is coincident with the base frame, and its z-axis is aligned with the gravity vector (*i.e.,* the roll and pitch orientations are perpendicular to the gravity vector).

3.3 Proprioceptive Sensors

In this section, we describe the characteristics of the proprioceptive sensors mounted on HyQ.

3.3.1 Encoders

Each one of the 12 joints of HyQ is equipped with a couple of absolute/relative, magnetic/optical encoders. The absolute encoder (AMS model AS5045) is a $12\,\mathrm{bit}$ digital magnetic encoder. With 4096 different outputs, its resolution is: $s_{\text{abs}} = 4096/360^\circ = 0.0879^\circ$. It is used only at the robot startup, to detect the initial position of the joints. After the robot startup and joint initialization with the absolute encoder, an incremental digital optical encoder (Avago model AEDA3300 BE1, see Figure 3.3a) is used. The relative encoder has 80000 counts per revolution, providing a resolution of: $s_{\text{rel}} = 80000/360^\circ = 0.0045^\circ$. The encoders are used to measure the joint positions $\mathbf{q} \in \mathbb{R}^{12}$, and to compute the joint velocities, $\dot{\mathbf{q}} \in \mathbb{R}^{12}$, through numerical differentiation.



(a) (b)

Figure 3.2: The Hydraulic Quadruped robot and its frames of reference. The Base frame is attached to the geometric center of the robot's torso and has the x-axis (red) pointing forward, the y-axis (green) pointing to the left, and the z-axis (blue) pointing upward; the IMU frame is also attached to the base link, with a different orientation; the World frame is a fixed reference frame.

(a) Avago AEDA3300 BE1 **(b)** Burster 8417

(c) Microstrain 3DM-GX3-25 **(d)** Microstrain 3DM-GX4-25 **(e)** KVH 1775

Figure 3.3: Proprioceptive sensors mounted on HyQ. **(a)** joint relative encoder; **(b)** loadcell; **(c)-(e)** IMUs.

3.3.2 Joint Force/Torque Sensor

Two of the three actuated joints of HyQ's leg, HFE and KFE, are moved by a piston. To measure the torque at these joints, we use Burster 8417 loadcells (Figure 3.3b), connected in series with the cylinders. From the force measured by the loadcell, we then compute the joint torque by multiplication with the lever arm's length. The measurement uncertainty for the loadcells is $\pm 25\,\mathrm{N}$. The torque at the HAA joint is instead directly measured by a custom made torque sensor (not depicted). In the following, we will indicate these sensors with the term *joint force/torque sensors*, and we implicitly assume that their output is always a torque $\boldsymbol{\tau} \in \mathbb{R}^{12}$.

3.3.3 Inertial Measurement Unit

Three different IMUs have been mounted on HyQ since its construction: the Microstrain 3DM-GX3-25 [91] (Figure 3.3c), the Microstrain 3DM-GX4-25 [92] (Figure 3.3d), and the KVH 1775 [68] (Figure 3.3e).

The Microstrain 3DM-GX3-25, and the newer model 3DM-GX4-25, are two compact industrial grade MEMS IMUs, commonly used in robotics, especially for drones. They internally run a Kalman-based filter, which provides an estimate of the device attitude, alongside the proper acceleration and angular velocity measurements. Even though there are many factors (and metrics) to consider when evaluating the performance of an IMU, one of the most critical ones is the gyro bias instability (see [147]). The GX3 and GX4 have a bias instability of $18\,^\circ/\mathrm{h}$ and $10\,^\circ/\mathrm{h}$, respectively. Thus, they are categorized as moderate performance grade [50] (*i.e.,* not suitable for navigation purposes by means of strapdown techniques only).

Since HyQ is designed to perform highly dynamic (yet precise) motions, the above mentioned IMUs have been accompanied with a higher grade one, the KVH 1775 [68] (Figure 3.3e). This sensor is characterized by navigation grade FOG gyros (0.05–$0.1\,^\circ/\mathrm{h}$ bias instability), which allows for a precise estimate of the attitude over longer periods of activity. During the DRC competition, 11 of the 23 team finalists (including the winner) relied on the performance of an almost identical device: the KVH 1750 IMU [1]. The same IMU is also adopted by the highly dynamic quadruped Cheetah [4].

3.4 Exteroceptive Sensors

In this section, we describe the exteroceptive sensors mounted on HyQ, including: depth sensors, stereo cameras, and LiDARs.

3.4.1 Depth Sensor

Depth sensor cameras, also known as RGB-D sensors, are a devices which output color point clouds by coupling a VGA camera, an Infrared (IR) camera, and a structured light projector. The projector illuminates the scene with a known light pattern, from which the device generates the

(a) ASUS Xtion

(b) Point Grey Bumblebee 2

(c) Hokuyo URG-04LX

(d) Velodyne HDL-32E

(e) Carnegie Robotics Multisense SL

Figure 3.4: Exteroceptive sensors used on HyQ. **(a)** depth sensor; **(b)** stereo camera; **(c)** planar LiDAR; **(d)** rotating LiDAR; **(e)** sensor head (LiDAR + stereo camera).

point cloud by triangulation. The point cloud is then colorized by projecting the image plane of the VGA camera onto the image plane of the IR camera.

The first notable consumer device of this kind, the Microsoft Kinect, was introduced in 2010. Since then, there has been an increasing interest in these commodity sensors, even in the robotics community. In particular, the ASUS Xtion sensor (Figure 3.4a) gained popularity due to its comparable performance yet smaller size and power consumption than the Kinect [54]. The ASUS Xtion mounted on HyQ has a 0.3 Megapixel rolling shutter camera, with a resolution of 640×480 pixel at 30 FPS, or 320×240 at 60 FPS. It is mainly used for indoor (due to sunlight interference) mapping and localization, as detailed in Chapter 5.

3.4.2 Stereo Camera

Due to the interference caused by sunlight, active depth sensors like the ASUS Xtion are not suitable for outdoor operations. Furthermore, these kind of sensors have a limited field of view (*cf.* Table 3.1). For this reason, HyQ is also equipped with stereo cameras. Two models of stereo cameras are used on HyQ: the Bumblebee 2 from Point Grey Inc. (Figure 3.4b), and the Multisense SL from Carnegie Robotics (Figure 3.4e).

The Bumblebee 2 mounted on HyQ is a global shutter 0.8 Megapixel camera, with maximum resolution of 1032×776 pixel at 20 FPS. It has been used mainly for mapping [56].

The Multisense SL (Figure 3.4e) is a tri-modal (LiDAR, camera, depth from stereo) sensor consisting of a rotating LiDAR, a stereo camera and integrated FPGA circuitry, to compute 3D point clouds from stereo, in hardware. The stereo camera of the Multisense SL used on HyQ has a 4 Megapixel imager, with a resolution of 1024×1024 pixel at 15 FPS. The sensor comes

pre-calibrated and synchronized for all the modalities. The camera is equipped with LED flash lights, to illuminate the scene when needed.

3.4.3 LiDAR

LiDAR sensors provide a strong support in obtaining high precision range measurements, even in conditions of scarce illumination, fog and dust. Three types of LiDAR are used on HyQ, mainly for localization purposes (see Chapter 5): the Hokuyo URG-04LX (Figure 3.4c), the Velodyne HDL-32E (Figure 3.4d), and the Hokuyo UTM-30LX-EW (Figure 3.4e).

The Hokuyo UTM-30LX-EW is a planar range scanner with a maximum range of $30\,\mathrm{m}$ and a wide Field of View (FoV) of $270°$. The scans are generated at $40\,\mathrm{Hz}$. As a component of the Multisense SL, it can fully rotate around a forward-looking axis of the device, to obtain a complete scan of the environment in a few seconds, keeping the device static. The sensor is dustproof and waterproof.

The Velodyne HDL-32E is a motorized LiDAR sensor, mainly produced for the automotive industry. Its bigger version, the HDL-64E, was one of the protagonists of the 2005 DARPA Grand Challenge and the 2007 DARPA Urban Challenge – two autonomous driving competitions organized by DARPA to push forward the research on self-driving ground vehicles. The sensor emits 32 beams, distributed vertically on a range of $40°$, and rotates freely around its vertical axis. This allows it to get a full point cloud, with $360°$ horizontal FoV, at $10\,\mathrm{Hz}$.

The Hokuyo URG-04LX is a short range ($4\,\mathrm{m}$), low-frequency ($10\,\mathrm{Hz}$) LiDAR sensor for indoor usage. It is rigidly attached to the robot's chassis (see Figure 3.5a) and it is used for 2D localization (see Section 5.3).

3.4.4 Depth Sensor Noise and Calibration

The noise of a point cloud highly depends on the used sensor. Table 3.1 shows an accuracy comparison, expressed in meters at different distances, between the Bumblebee 2, the ASUS Xtion and the Multisense SL's stereo camera. Even though it is more reliable than a stereo camera, when the distance approaches a few meters, the accuracy drop starts becoming non-negligible. To cancel the noise, a few preprocessing methods can be used. For example, [97] recalibrated the structured light sensors to improve the depth image quality and [139] corrected depth distortion.

3.5 Sensor Configurations

From its construction in 2010, the HyQ robot received a few mechanical and hydraulic upgrades, which included the removal of the passive springs from the lower leg, and the substitution of the electric motors with hydraulic rotary actuators at the HAA joints. More recently, also the sensory system has evolved towards better performing, integrated, and rugged solutions. In the following sections, we describe the two sensor setups used for the experiments and the dataset collection described in the following chapters. We will denote them with the terms

Specification	Bumblebee 2	ASUS Xtion	Multisense SL
Output [Megapixel]	0.8	0.3	4
Frequency [FPS]	20	30	15
Depth resolution [m@m]	0.01 @ 1.29 0.5 @ 9.6 1 @ 13.6	0.001 @ 0.8 0.04 @ 4	± 0.0003 @ 1 ± 0.03 @ 10
Range [m]	0.6–20	0.8–4	0.4–10
Valid depth [%]	65%	84%	N/A
HFOV×VFOV [°]	97×72	57×40	80×80

Table 3.1: Comparison between depth sensors specification used on HyQ: the Bumblebee 2 stereo camera, the structured light ASUS Xtion, and the Multisense SL stereo camera.

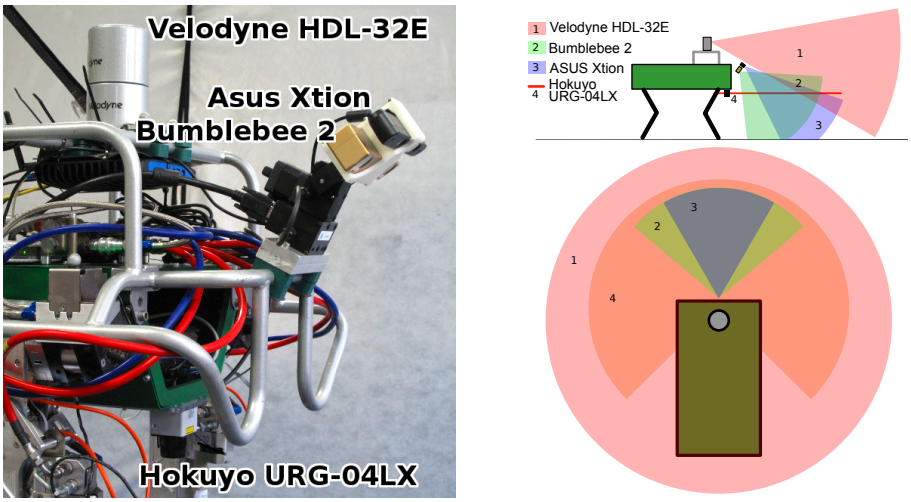
Sensor	Configuration A	Configuration B
ASUS Xtion	✓	✓
Bumblebee 2	✓	
Velodyne HDL-32E	✓	
Hokuyo URG-04LX	✓	✓
FLIR PTU-D46-17	✓	
Multisense SL		✓
Microstrain 3DM-GX3-25	✓	
Microstrain 3DM-GX4-25		✓
KVH 1775		✓

Table 3.2: List of sensors and additional actuators for Configuration A and Configuration B.

Configuration A and *Configuration B*. Both configurations share the same encoder and joint force/torque sensor setup, while the other sensors change from one to the other. Table 3.2 summarizes the sensors available for each configuration.

3.5.1 Configuration A

In Configuration A, HyQ is equipped with a 3DM-GX3-25 IMU (Figure 3.3c), rigidly attached to the robot base. The exteroceptive sensors are visible on Figure 3.5a: a Bumblebee 2 stereo camera (Figure 3.4b), and an ASUS Xtion depth sensor (Figure 3.4a), are mounted on a Pan-and-Tilt Unit (PTU), which provides two degrees of freedom for active scanning [90] and image stabilization purposes [12]. The Hokuyo URG-04LX is placed below the torso, upside-down, and the Velodyne HDL-32E is mounted on top of the robot. A schematic of LiDAR's and camera's FoVs is provided



(a) Sensor setup. **(b)** Field of View scheme.

Figure 3.5: Exteroceptive sensors in Configuration A and their Fields of View. Note that the fields of view are not in scale and do not consider self-occlusions.

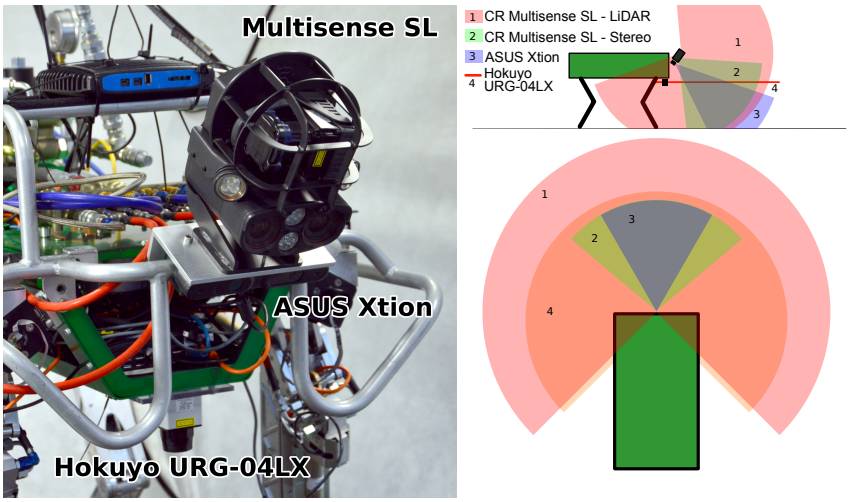
in Figure 3.5b.

This configuration has been used for the experiments presented in Section 5.2, Section 5.3, and Section 5.4. The datalogs of Dataset 1 (Appendix A.1) and Dataset 2 (Appendix A.2) were also recorded in this configuration.

3.5.2 Configuration B

In Configuration B, the Bumblebee 2 stereo camera, the Pan-and-Tilt-Unit, and the Velodyne LiDAR are replaced by the Multisense SL (Figure 3.6a). This choice is motivated by multiple factors: the stereo camera from the Multisense SL has a higher resolution and provides point clouds directly; the rotational axis of the LiDAR permits the use of laser returns for terrain mapping, as well as for localization; all the sensor modalities are synchronized via hardware, with no extra effort from the user (see Section 3.7); the Multisense SL device is rugged and tested against vibration and high temperatures. The Microstrain 3DM-GX3-25 is replaced by its upgraded model, the 3DM-GX4-25 (Figure 3.3d). The configuration also includes a KVH 1775 FOG IMU (Figure 3.3e). Figure 3.6 shows the sensor arrangement for this configuration and the corresponding FoV of the exteroceptive sensors. Note that, with this new LiDAR setup, the rear view is lost, but there is more scanning surface in front of the robot.

The Configuration B is used in the experimental session of Section 5.5, as well as in the dataset of Appendix A.3.



(a) Sensor setup. **(b)** Field of View scheme.

Figure 3.6: Exteroceptive sensors in Configuration B and their Fields of View. Note that the fields of view are not in scale and do not consider self-occlusions.

3.6 Camera Pose Calibration

One of the critical aspects of state estimation with exteroceptive sensors is the availability of the correct rigid transformation between these sensors and the base frame of the robot. For instance, if we perform VO with a camera, we need to correctly estimate the rigid transformation ${}_b\mathrm{T}_c$ which maps the points in the camera frame c to the base frame b. Retrieving this transform is not straightforward, because of:

1. mechanical tolerances between assembly parts of the robot;
2. mechanical tolerances and distortion of the depth sensors;
3. non-rigidity of the materials, which produce undesired motions when shocks occur.

While issue 3) is impossible to solve in practice, issues 1) and 2) can be solved trough an appropriate static calibration procedure. In order to retrieve a valid transformation ${}_b\mathrm{T}_c$, we have developed a calibration method which uses a motion capture system and a library for augmented reality, ArUco [45]. In the following, we show how to perform the calibration with Bumblebee 2, from Configuration A. However, the method can be generalized to any stereo camera.

Figure 3.7 depicts the calibration setup: a fiducial marker is placed on the floor, within the FoV of the stereo camera. The fiducial marker is surrounded by motion capture markers (small reflective dots). Without loss of generality, we assume that the frame origin of the fiducial marker and the origin defined by the motion capture polygon are the same. We indicate this unified marker frame as m. The rigid transformation ${}_b\mathrm{T}_c$ is then computed as:

$$ {}_b\mathrm{T}_c = \left({}_w\mathrm{T}_b\right)^{-1} {}_w\mathrm{T}_m \left({}_c\mathrm{T}_m\right)^{-1} \tag{3.1} $$

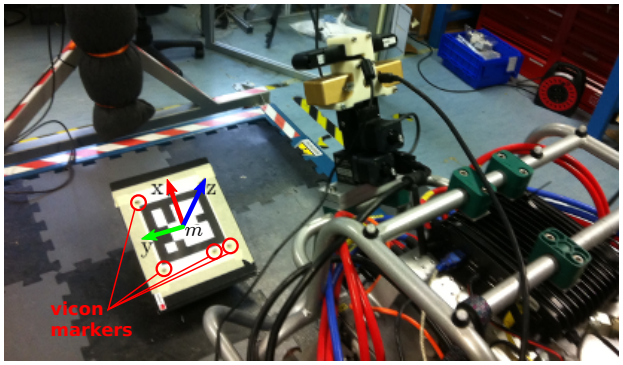


Figure 3.7: Sensor calibration setup. A fiducial marker is surrounded by motion capture markers. The transform between optical frame and the marker frame is computed on the 3D points extracted from the fiducial marker. The rest of the transformation chain is extracted via motion capture system.

where ${}_wT_b$ and ${}_wT_m$ are the rigid transformations (provided by the motion capture system) that project the base frame and the marker frame onto the world frame, respectively. The rigid transformation from the marker frame m to the camera optical frame c is defined as ${}_c\mathrm{T}_m$. Let $\mathbf{t}$ be the translational component of ${}_c\mathrm{T}_m$. Its value corresponds to the vector coordinates of the marker center, expressed in the camera frame. The rotational part of ${}_c\mathrm{T}_m$ can instead be computed through SVD, as the solution to Wahba's problem [89]:

$$ {}_mT_c = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} UMV^{\mathrm{T}} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{3.2} $$

where M, U and V are extracted from:

$$ B = \sum_{i=1}^{n} \mathbf{w}_i \mathbf{v}_i^{\mathrm{T}} = USV^{\mathrm{T}} \tag{3.3} $$

$$ M = \mathrm{diag}\begin{bmatrix} 1 & 1 & \det(U)\det(V) \end{bmatrix} \tag{3.4} $$

and $\mathbf{w}_i$, $\mathbf{v}_i$ are the sets of corner points on the marker, expressed in the marker frame m and the camera frame c, respectively.

3.7 Synchronization

As we have seen in the previous sections, HyQ is a complex robot. It is equipped with several sensors, each one providing signals at different frequencies, bandwidths, and from different interfaces. These signals eventually have to be fused, in order to generate a state estimate of the robot base. For this, we need a common time frame and a synchronization mechanism. The importance of synchronization between heterogeneous sensor samples is confirmed by the need, from several groups, to solve this problem via hardware with custom FPGA boards [128, 99, 25, 83]. Although this is the most effective solution, it is not applicable anywhere on

the robot. In our case, the Multisense SL provides this capability for synchronizing the LiDAR and the stereo camera images. For the other signals, we rely on the following strategies:

- the joint states (encoders and joint force/torque sensors) signals are synchronized through EtherCAT [2] at 1 kHz in a realtime environment;
- the signals generated by the IMU, which is directly connected to the controller computer, are routed with the Lightweight Communication and Marshalling (LCM) library [63] (a inter-process communication library, suitable for low latency robotic applications). The synchronization with the other signals is performed using the absolute timestamp of the computer, fused with the local timestamp of the device, with a passive technique [102];
- synchronization between different computers is operated through the Network Time Protocol [3].

3.8 Locomotion and Perception Capabilities

HyQ is designed to be a versatile robot. Its mechanical and hydraulic structure is able to support a variety of gaits and maneuvers, including: trotting [8], static crawling [146], dynamic crawling [145], climbing on inclined surfaces [43], and bounding [103]. Over the course of its development at the Dynamic Legged System Lab, the progressive integration between control and perception allowed for an increasing number of sophisticated locomotion skills.

In 2013, Focchi *et al.* [42] demonstrated that, by monitoring the joint force/torque sensors and estimating the impact forces at the feet, the periodic trajectories generated by the Reactive Controller Framework (RCF) [8] can be reactively adjusted to step over an unexpected obstacle of 11 cm, which corresponds to 14 % of the total leg length. In the same year, Havoutis *et al.* [56] showed, experimentally on HyQ, the first implementation of an onboard perception-assisted gait transition between a dynamic trot and a static crawl. Their proposed method consists of the following steps: 1) point cloud acquisition from a depth camera; 2) dominant plane extraction, to determine the terrain inclination, which is assumed to be flat; 3) computation of the robot orientation from the extracted plane orientation and IMU; 4) computation of the robot position, from image registration techniques [113]; 5) generation of elevation maps, from the point clouds and estimated robot pose. From the elevation map analysis, the robot was able to switch from a trotting to a crawling gait, and step on a 14 cm pallet. In 2014, Bazeille *et al.* showed a trotting gait parameter modulation assisted by stereo imagery. Velocity, step height, and length were adjusted according to the characteristics of the point cloud captured by the robot. In the same year, Winkler *et al.* [146] presented a framework for foothold planning, with force-based foothold adaptation, to overcome highly challenging terrain. The planned footholds were executed on a pre-acquired map with the KinectFusion algorithm [96]. In 2015, Winkler *et al.* [145] extended their previous work with a dynamically stable crawl. The maps were in this case pre-acquired with a depth sensor and collected before the execution, in an OctoMap data structure [61].

In the following chapters, we describe in detail the fundamental capabilities (state estimation, localization, mapping) required to migrate these and other locomotion skills from controlled lab environments towards real scenarios.

Chapter 4

Probabilistic Contact Estimation for Proprioceptive State Estimation

This chapter addresses the problem of estimating the 6-DoF position and velocity of the HyQ robot's base with proprioceptive sensors. In particular, we want to estimate the base state using an IMU, joint encoders, and joint force/torque sensors. IMUs provide direct measurements of the base angular velocity and linear acceleration. The encoders are generally used to compute the base linear velocity [16] or position [17] from forward kinematics (see Section 2.1.2). To produce valid measurements, only the feet firmly in contact with the ground have to be considered. For this reason, most state estimators in legged robotics [17, 16, 37, 84] use dedicated contact sensors to detect which feet are able to provide reliable velocity or position estimates.

In this chapter, we explain how to identify the feet in reliable contact with the ground by using joint force/torque sensors, instead of contact sensors. Avoiding the use of contact sensors at the feet (especially on machines the size of HyQ) is desirable for many reasons: it reduces the overall cost, weight, and complexity of the system [67]; it allows the use of rugged, inexpensive and easily replaceable feet; it reduces the number of single points of failure (because dependency on contact sensors compromises the functionality of the overall system in case any of them are damaged).

Without dedicated contact sensors, detecting and handling contacts is not trivial, as they depend on the amount of frictional force the feet exert on the terrain [123]. According to the Coulomb model of dry friction, this is directly proportional to the normal component of the Ground Reaction Force (GRF) and the foot-terrain static friction coefficient, which is generally unknown. Additionally, impact forces play a critical role, as they can cause slippage [20]. Throughout this chapter, we describe a method to probabilistically detect the contact threshold on the GRF that minimizes the error of the velocity measurements obtained from the joint encoders and forward kinematics. We also describe how to use this information to fuse multiple velocity contributions from the stance legs into one measurement for the Kalman measurement update step. These methods have been validated on a dataset of more than one hour of locomotion logs (see Appendix A.2) on the HyQ robot.

The remainder of this chapter is structured as follows: Section 4.1 formally defines the base state estimation problem and describes the EKF framework implemented on HyQ (which is used also in Chapter 5); Section 4.2 describes a novel probabilistic approach to contact estimation for state estimation and compares its performance with other state-of-the-art approaches found in literature; Sections 4.3 and 4.4 shows in detail how the linear velocity measurement and its associated covariance are computed from the detected contact and integrated into the filter; Section 4.5 show an extensive performance validation of the state estimation framework method with experiments on HyQ performing a variety of gaits; Section 4.6 discusses the limitation of the approach and possible improvements; and finally, Section 4.7 summarizes the content of the chapter.
Most of the material presented in this chapter has been published in [27] as first author.

4.1 Filter Framework

We define the robot base state vector to be estimated as:

$$\boldsymbol{x} = \begin{bmatrix} {}_w\mathbf{x}_b & {}_b\dot{\mathbf{x}}_b & {}_b\ddot{\mathbf{x}}_b & {}_w\Theta_b & {}_b\boldsymbol{\omega}_b & \mathbf{b}_a & \mathbf{b}_\omega \end{bmatrix}^{\mathrm{T}} \tag{4.1}$$

where the base velocity ${}_b\dot{\mathbf{x}}_b \in \mathbb{R}^3$, acceleration ${}_b\ddot{\mathbf{x}}_b \in \mathbb{R}^3$ and rotational rate ${}_b\boldsymbol{\omega}_b \in \mathbb{R}^3$ are expressed in the base frame b, while the position ${}_w\mathbf{x}_b \in \mathbb{R}^3$ and orientation ${}_w\Theta_b \in \mathrm{SO}(3)$ are expressed in the fixed world frame w (for the frames of reference and their locations, see Section 3.2). Finally, the state space is completed by IMU acceleration and angular velocity biases: $\mathbf{b}_a \in \mathbb{R}^3,\ \mathbf{b}_\omega \in \mathbb{R}^3$.

To estimate the state, we adopt the EKF framework of [22, 37]. The goal of the EKF is to estimate the mean and covariance of the Gaussian distribution over the state $\boldsymbol{x}_k$ at time k (see Appendix B.2). The prior distribution over the state is propagated using the acceleration and angular velocity sensed by the IMU as inputs, while the posterior is subsequently computed by integration of velocity measurements from the LO.

4.1.1 Inertial Process Model

An IMU measures proper acceleration and angular velocity. Given the rotation matrix $R_b^i \in \mathrm{SO}(3)$ and translation vector $\mathbf{t}_i^b \in \mathbb{R}^3$, which describe the rigid transform from IMU frame to base frame, we can compute the measurements of the base acceleration ${}_b\ddot{\mathbf{x}}_b$ and angular velocity ${}_b\boldsymbol{\omega}_b$, as follows [32]:

$${}_b\boldsymbol{\omega}_b = R_i^b({}_i\boldsymbol{\omega}_b - \mathbf{b}_\omega) = R_i^b({}_i\boldsymbol{\omega}_i - \mathbf{b}_\omega) \tag{4.2}$$

$${}_b\ddot{\mathbf{x}}_b = R_i^b({}_i\ddot{\mathbf{x}}_i - \mathbf{b}_a) - {}_b\mathbf{g} \underbrace{- (R_i^b\, {}_i\dot{\boldsymbol{\omega}}_i) \times \mathbf{t}_i^b}_{\text{angular acceleration}} \underbrace{- (R_i^b\, {}_i\boldsymbol{\omega}_i) \times \left[(R_i^b\, {}_i\boldsymbol{\omega}_i) \times \mathbf{t}_i^b \right]}_{\text{centripetal force}} \tag{4.3}$$

Note that since the IMU is sensing the proper acceleration, we need to subtract the gravity vector from the measurement, expressed in the base frame ${}_b\mathbf{g} = R^b_{w\,w}\mathbf{g} = R^b_w \begin{bmatrix} 0 & 0 & 9.80655 \end{bmatrix}^{\mathrm{T}}$. This requires the knowledge of the current robot orientation R^b_w.

Prior State

Following commonly-used approaches, we use acceleration and angular velocity from Equations 4.3 and 4.2 as filter inputs $\boldsymbol{u}$ and remove them from the state $\boldsymbol{x}$. For readability, we drop the frame subscripts, add the time subscripts, and mark the prior and posterior states with the $-$ and $+$ superscripts. The input $\boldsymbol{u}_k$ and prior state $\boldsymbol{x}^-_k$ are defined as:

$$\boldsymbol{u}_k = \begin{bmatrix} \tilde{\boldsymbol{\omega}}_k \\ \tilde{\ddot{\mathbf{x}}}_k \end{bmatrix} = \begin{bmatrix} R^b_i({}_i\boldsymbol{\omega}_i - \mathbf{b}^+_{\omega,k-1}) \\ R^b_i({}_i\ddot{\mathbf{x}}_i - \mathbf{b}^+_{a,k-1}) \end{bmatrix} \tag{4.4}$$

$$\boldsymbol{x}^-_k = \begin{bmatrix} \mathbf{x}^-_k \\ \dot{\mathbf{x}}^-_k \\ \Theta^-_k \\ {}_a\mathbf{b}^-_k \\ {}_\omega\mathbf{b}^-_k \end{bmatrix} = \begin{bmatrix} \mathbf{x}^+_{k-1} \\ \dot{\mathbf{x}}^+_{k-1} \\ \mathbf{0} \\ {}_a\mathbf{b}^+_{k-1} \\ {}_\omega\mathbf{b}^+_{k-1} \end{bmatrix} + \begin{bmatrix} \dot{\mathbf{x}}^+_{k-1}\Delta t \\ (-\tilde{\boldsymbol{\omega}}_k \times \dot{\mathbf{x}}^+_{k-1} + (\Theta^+_{k-1})^{-1}{}_w\mathbf{g} + \tilde{\ddot{\mathbf{x}}}_k)\Delta t \\ \Theta^+_{k-1}\exp(\boldsymbol{\omega}^\wedge_k \Delta t) \\ \dot{\mathbf{b}}^-_{a,k-1}\Delta t \\ \dot{\mathbf{b}}^-_{\omega,k-1}\Delta t \end{bmatrix} \tag{4.5}$$

In Equation 4.4 we assume the effects of the angular acceleration and the centripetal force of Equation 4.3 to be negligible. Biases $\mathbf{b}_a$ and $\mathbf{b}_\omega$ are computed at the filter initialization, when the robot is stationary. At runtime, the corresponding bias rates are treated as white Gaussian noise, and integrated accordingly [147]. Noise parameters for bias rates can be extracted from datasheets, from the analysis of Allen variances [34], or by process identification [22].

Equation 4.5 computes the prior state using Euler integration over the timestep Δt: the current position $\mathbf{x}^-_k$ is estimated from the previous posterior position and velocity; current velocity $\dot{\mathbf{x}}^-_k$ depends on the previous states and IMU inputs (here indicated with a $\sim$, to distinguish them from the quantities in Equations 4.2 and 4.3). Note that the previous orientation Θ^+_{k-1} is required to align the gravity vector to the base frame and to compute the current orientation Θ^-_k from Equation 2.6.

Prior Covariance

The prediction step of the EKF is completed by the computation of the covariance P^-_k, which is propagated as follows:

$$P^-_k = G_k P^+_{k-1} G^{\mathrm{T}}_k + V_k R_k V^{\mathrm{T}}_k \tag{4.6}$$

The matrices $G_k = I + G_c\Delta t$, and $V_k = V_c\Delta t$, are computed from the partial derivatives of the nonlinear transition function $\boldsymbol{x}_{k+1} = g(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k)$ over the state and the input vectors,

respectively [22]:

$$G_c = \frac{\partial g}{\partial \boldsymbol{x}} = \begin{bmatrix} -\boldsymbol{\omega}^\wedge & (\Theta^{\mathrm{T}}\mathbf{g})^\wedge & 0 \\ 0 & -\boldsymbol{\omega}^\wedge & 0 \\ \Theta & -\Theta\dot{\mathbf{x}}^\wedge & 0 \end{bmatrix} \tag{4.7}$$

$$V_c = \frac{\partial g}{\partial \boldsymbol{u}} = \begin{bmatrix} \dot{\mathbf{x}}^\wedge & \mathbf{g}I \\ I & 0 \\ 0 & 0 \end{bmatrix} \tag{4.8}$$

4.1.2 Measurement Model

Let us consider the joint position and velocity measurements: $\mathbf{q} \in \mathbb{R}^n, \dot{\mathbf{q}} \in \mathbb{R}^n$, with $n = 12$ for HyQ. Thus, the velocity of each foot f_l, expressed in the base frame, can be computed through forward kinematics, as:

$${}_b\dot{\mathbf{x}}_{f_l} = J_l(\mathbf{q})\dot{\mathbf{q}}_l \tag{4.9}$$

where $J_l(\mathbf{q}) \in \mathbb{R}^{3\times3}$ is the Jacobian which maps the joint angles of leg l to the end effector (*i.e.,* the foot), and $\dot{\mathbf{q}}_l \in \mathbb{R}^3$ is the block of the joint state vector relative to leg l. If the leg l is stationary with respect to the ground, a measure of the base velocity can be computed from Equation 4.9 as follows:

$${}_b\dot{\mathbf{x}}_{b_l} = -{}_b\dot{\mathbf{x}}_{f_l} - {}_b\boldsymbol{\omega}_b \times {}_b\mathbf{x}_{f_l}, \tag{4.10}$$

where ${}_b\boldsymbol{\omega}_b$ is computed from Equation 4.2 and ${}_b\mathbf{x}_{f_l}$ is the position of foot f_l expressed in the base frame.

From the contributions ${}_b\dot{\mathbf{x}}_{b_l}, \forall l \in L = \{\mathrm{LF}, \mathrm{RF}, \mathrm{LH}, \mathrm{RH}\}$ (see Section 3.2) we can compute a Kalman measurement $\dot{\mathbf{x}}_k$. To perform this operation, we need: a) to know which feet are in stable contact with the ground; b) to fuse the individual leg contributions into a single EKF measurement update; c) to compute the associated covariance. In the following sections we cover these three points.

4.2 Probabilistic Contact Estimation

In this section, we describe a contact estimation method suitable for base state estimation. Generally, contact estimation is a relevant problem for Human-Robot interaction, where the robot has to detect a contact or a collision as early as possible, in order to promptly take counter-measures to avoid damage to either a human or the machine [82, 53]. In contrast, the method described in this section is designed for state estimation, and it aims to find the force threshold at which the foot deemed to be in contact can provide reliable velocity measurements for the EKF. The two time instances when this threshold is crossed and when a contact event begins might be different, with the latter typically happening earlier.

4.2.1 Ground Reaction Forces Estimation

We estimate the GRFs by considering the equation of motion for a floating base system. The dynamics of a floating-base articulated-body system can be expressed as two coupled dynamics equations, computed using Recursive Newton-Euler algorithms, as described in [39]. The first equation describes the dynamics of the floating-base body (6 DoFs, underactuated), while the second one describes the dynamics of the n rigid-bodies (*i.e.,* for HyQ, $n = 12$) attached to it through active joints (*i.e.,* active DoFs). The equations of motion of such a system can be partitioned as follows:

$$\left[\begin{array}{c:c} I_c & F \\ \hdashline F^T & M \end{array}\right] \begin{bmatrix} {}_b\ddot{\mathbf{x}}_b \\ {}_b\dot{\boldsymbol{\omega}}_b \\ \hdashline \ddot{\mathbf{q}} \end{bmatrix} + \begin{bmatrix} \mathbf{h}_b \\ \hdashline \mathbf{h}_q \end{bmatrix} = \begin{bmatrix} J_{cb}^T \\ \hdashline J_{cq}^T \end{bmatrix} \mathbf{f} + \begin{bmatrix} \mathbf{0} \\ \hdashline \boldsymbol{\tau} \end{bmatrix} \tag{4.11}$$

where: $I_c \in \mathbb{R}^{6\times6}$ is the composite rigid body inertia of the robot; $F \in \mathbb{R}^{6\times n}$ is a matrix which contains the spatial forces required at the floating base to support unit accelerations about each joint variable; h_b is the spatial bias force (*e.g.,* gravity/centrifugal), for the composite rigid body containing the whole floating-base system; $M \in \mathbb{R}^{n\times n}$ denotes the active joint link's inertia matrix (*i.e.,* leg segments); $h_q \in \mathbb{R}^n$ denotes the respective vector of Coriolis, centrifugal and gravitational forces; $\ddot{\mathbf{x}}_b \in \mathbb{R}^3$ and $\dot{\boldsymbol{\omega}} \in \mathbb{R}^3$ are the vectors representing the floating-base linear and angular accelerations, respectively; $\ddot{q} \in \mathbb{R}^n$ and $\tau \in \mathbb{R}^n$ are the vectors of active joints accelerations and torques. If c is the number of contacts with the ground, $\mathbf{f} \in \mathbb{R}^{3c}$ is the vector of GRFs that enter in the dynamic equation through the contact Jacobians $J_{cb} \in \mathbb{R}^{3c\times6}$ and $J_{cq} \in \mathbb{R}^{3c\times n}$. Note that since our platform has nearly point-like feet, we assume that it cannot generate moments at the contact points – but only pure forces (hence, 3 components for each leg). Since the mass of each leg of HyQ is less than 8% of the total robot weight, we opt to neglect the effect of inertial torques. This avoids introducing noise in the estimation, due to the numerical differentiation in the computation of $\ddot{\mathbf{q}}$. Therefore, we can estimate the GRFs from Equation 4.11, as:

$$\mathbf{f} = -(J_{cq}^T(\mathbf{q}))^\dagger \left(\boldsymbol{\tau} - \mathbf{h}_q - F^{\mathrm{T}} \begin{bmatrix} {}_b\ddot{\mathbf{x}}_b & {}_b\dot{\boldsymbol{\omega}}_b \end{bmatrix}^{\mathrm{T}} \right) \tag{4.12}$$

where $(\cdot)^\dagger$ is the Moore-Penrose pseudo-inverse. In practical applications, due to the block-wise structure of J_{cq}, it is possible to perform the computation for each stance leg separately as:

$$\mathbf{f}_l = -(J_{cq_l}^{\mathrm{T}})^{-1} \left(\boldsymbol{\tau}_l - \mathbf{h}_{q,l} - F_l^{\mathrm{T}} \begin{bmatrix} {}_b\ddot{\mathbf{x}}_b & {}_b\dot{\boldsymbol{\omega}}_b \end{bmatrix}^{\mathrm{T}} \right) \tag{4.13}$$

where: $\mathbf{f}_l \in \mathbb{R}^3$ is the GRF for leg l; $\mathbf{q}_l \in \mathbb{R}^3$, $\dot{\mathbf{q}}_l \in \mathbb{R}^3$ and $\boldsymbol{\tau}_l \in \mathbb{R}^3$ are the leg joint position, velocity and torque, respectively; $J_{cq_l}(\mathbf{q}_l)$ is the l-th foot Jacobian (which for HyQ is a square matrix); $\mathbf{h}_{q,l}$ is the vector of centrifugal/Coriolis/gravity torques, for leg l.

4.2.2 Contact Classification

We define the contact status for a foot belonging to leg $l \in \{\text{LF, RF, LH, RH}\}$ as $s_l \in \{0, 1\}$, where 1 indicates a reliable stance (*i.e.,* with no motion relative to the ground) and 0 indicates leg swing phase or slipping contact.

Given $\mathbf{f}_l = (f_{l,x}, f_{l,y}, f_{l,z})$ and following the definition of [59], the quantity:

$$\mu_f = \frac{\sqrt{f_{l,x}^2 + f_{l,y}^2}}{f_{l,z}}, \quad \forall f_{l,z} > 0 \tag{4.14}$$

defines a metric to evaluate the robustness of a foothold in terms of contact stability. This metric is equal to the actual static friction coefficient μ_s when the lateral components of the GRF, denoted with $f_{l,x}, f_{l,y}$, have a value beyond which the foot would start slipping. Although μ_s is unknown, any value of $\mu_f < \mu_s$ would yield a stable contact, and in particular, the smaller μ_f is, the more likely the foot is firmly on the ground. Hence, the quality of contact for a foot related to leg l at time k is nonlinearly proportional to the corresponding GRF $\mathbf{f}_l^k$. For simplicity and numerical stability, instead of accounting for all the terms of μ_f, we ignore the lateral components of $\mathbf{f}_l^k$, and assume that, above a certain threshold of $f_{l,z}^k$, the frictional force will be sufficient to produce a stable, reliable contact. To learn this threshold, we model the probability of a reliable ground contact $\mathcal{P}_k$ using a discriminative logistic regression model:

$$\mathcal{P}_k(s_l = 1 | \mathbf{f}_l^k) = \frac{1}{1 + \exp(-\beta f_{z,l}^k - \beta_0)} \tag{4.15}$$

where $f_{z,l}^k$ is the normal component of the GRF at time k for leg l, while β and β_0 can be regarded as the weights of a logistic regression classifier. The weights are computed by maximum likelihood estimation on a training set of data collected from characteristic locomotion behaviors, as described in the following section.

4.2.3 Training Set Generation

To generalize their applicability, logistic classifiers require a good training set, sufficiently large and able to cover the region of interest of the input. In our case, the training set includes the ground truth of contact events $\mathbf{s}_k = (s_{\text{LF}}, s_{\text{RF}}, s_{\text{LH}}, s_{\text{RH}})_k$, and the corresponding GRF values for all the legs. Excluding the data generated by simulation, we assume that the ground truth for contact events is not available. Using manually labeled data is impractical for datasets longer than a few minutes, therefore we need to define a semi-supervised routine to generate them.

Let us consider that for every time step k and leg $l \in L$, the corresponding velocity contribution $\hat{\dot{\mathbf{x}}}_{k,l}$ is computed from Equation 4.10. We also assume to have access to the velocity ground truth $\dot{\mathbf{x}}_k$ for the base. Thus, for each time step k, we can compute the optimal leg

combination $\mathbf{s}^* \in \mathbb{B}^4$, which minimizes the absolute velocity error:

$$\mathbf{s}_k^* = \underset{\mathbf{s}\in\mathbb{B}^4}{\arg\min} \left| \left[\frac{1}{\bar{s}} \sum_{l\in L} \mathrm{I}(\hat{\mathbf{x}}_{k,l}, s_l) \right] - \dot{\mathbf{x}}_k \right| \quad \forall k > 0 \tag{4.16}$$

where:

- $\mathrm{I}(a, b)$ is the indicator function, which outputs 0 if $b = 0$ and a otherwise;
- s_l is the l-th element of $\mathbf{s}$;
- $\bar{s}$ is the sum of non-zero elements of $\mathbf{s}$.

Assuming that error is lower for a firmer contact, Equation 4.16 provides a basic rule to generate a training set which teaches the classifier the optimal threshold on the GRF signals for triggering the contact. However, in a practical implementation, we have to take into account the special case of a robot flying phase (*i.e.,* , when $\mathbf{s} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$). We also have to consider that numerical instability in the optimization process (*e.g.,* two leg combinations giving very similar errors) may lead to unrealistic frequent contact phase switching.

Starting from Equation 4.16, the complete algorithm to generate the training set is detailed in Algorithm 1. For each timestep, k and leg combination, $\mathbf{s}$ we compute the average base velocity (lines 7–11) of the 14 possible leg combinations (all except the "no legs in stance" combination). Then we choose the combination which minimizes the error against the ground truth, E (lines 14–16). The flying phase combination is treated separately: if all the non-flying combinations provide an error beyond a threshold ϵ, the flying phase is triggered (line 19).

After computing the optimal sequence $\mathbf{s}^*$ for every timestep k, we manually fill the gaps and remove spurious contacts (line 24). The error threshold ϵ and the parameters for the region growing algorithm are manually selected.

Algorithm 1 Contact Training Set Generation Algorithm

for each $k > 0$ **do**
  $E^* \leftarrow \infty$ ▷ initialize minimum error
  $\mathbf{s}^* \leftarrow (0, 0, 0, 0)$ ▷ initialize leg combination
  for each $\mathbf{s} \in \mathbb{B}^4 - \{(0, 0, 0, 0)\}$ **do**
    $\bar{s} \leftarrow 0$ ▷ initialize number of stance legs
    $\hat{\dot{\mathbf{x}}}_k = (0, 0, 0)$ ▷ initialize estimated velocity
    for $l = 1 : 4$ **do**
      $\hat{\dot{\mathbf{x}}}_{k,l} = -\dot{\mathbf{x}}_{k,f_l} - \boldsymbol{\omega}_k \times \mathbf{x}_{k,f_l}$
      $\hat{\dot{\mathbf{x}}}_k \leftarrow \hat{\dot{\mathbf{x}}}_k + \mathrm{I}(\hat{\dot{\mathbf{x}}}_{k,l}, s_l)$
      $\bar{s} \leftarrow \bar{s} + \mathrm{I}(1, s_l)$
    end for
    $\hat{\dot{\mathbf{x}}}_k \leftarrow \hat{\dot{\mathbf{x}}}_k / \bar{s}$
    $E \leftarrow |\hat{\dot{\mathbf{x}}}_k - \dot{\mathbf{x}}|$ ▷ compute error
    if $E < E^*$ **then**
      $E^* \leftarrow E$
      $\mathbf{s}^* \leftarrow \mathbf{s}$
    end if
  end for
  if $E^* > \epsilon$ **then**
    $\mathbf{s}^* = (0, 0, 0, 0)$
  end if
  $\mathbf{s}_k \leftarrow \mathbf{s}^*$
end for
$\mathrm{filter}(\mathbf{s}_1, \ldots, \mathbf{s}_k, \ldots)$

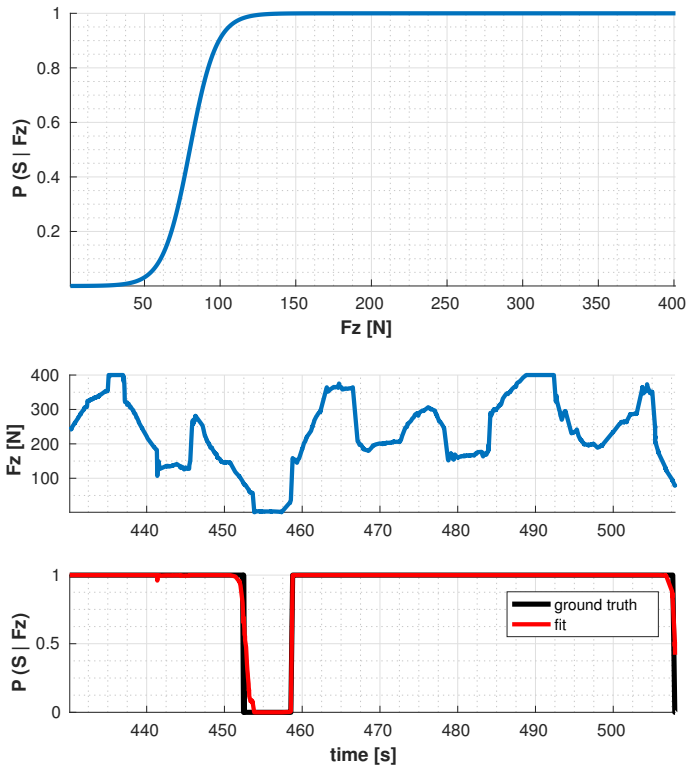


Figure 4.1: Crawl gait simulation. *Top plot:* learned logistic model function. *Middle plot:* normal component of the GRF for one leg. *Bottom plot:* learned stance probability and ground truth

4.2.4 Performance Evaluation

In this section we evaluate the performance of the contact detection algorithm. A comparison with other common methods is shown for both simulation and real data. Since the target application for the contact estimation is state estimation, the performance metric of choice is the Drift per Distance Traveled (DDT) (see Appendix C).

Fitting with Simulated Data

First, we tested our approach on data generated from simulation, with contact ground truth, for two distinctive locomotion styles: a static crawl and a dynamic trot. The crawl gait was generated using a quasi-static crawl controller, as described in [43]. The trot gait was generated using the reactive controller framework presented in [8]. In our experiments, the trot gait was generated using a step frequency of $1.7\,\mathrm{Hz}$, a duty factor of 0.5 and a leg stiffness of $8.55 \times 10^3\,\mathrm{N/m}$. All simulation are performed on flat ground.

Figures 4.1 and 4.2 show — for crawl and trot datalogs, respectively — the learned logistic function (top plot), the GRF signal (middle plot), and the fitting of the model against the ground truth for the test set (bottom plot). As expected, the threshold for contact activation in the trot gait is higher (by approx. $20\,\mathrm{N}$). This is due to the fact that for this locomotion gait two legs are off the ground at a time, compared to just one in the crawl.

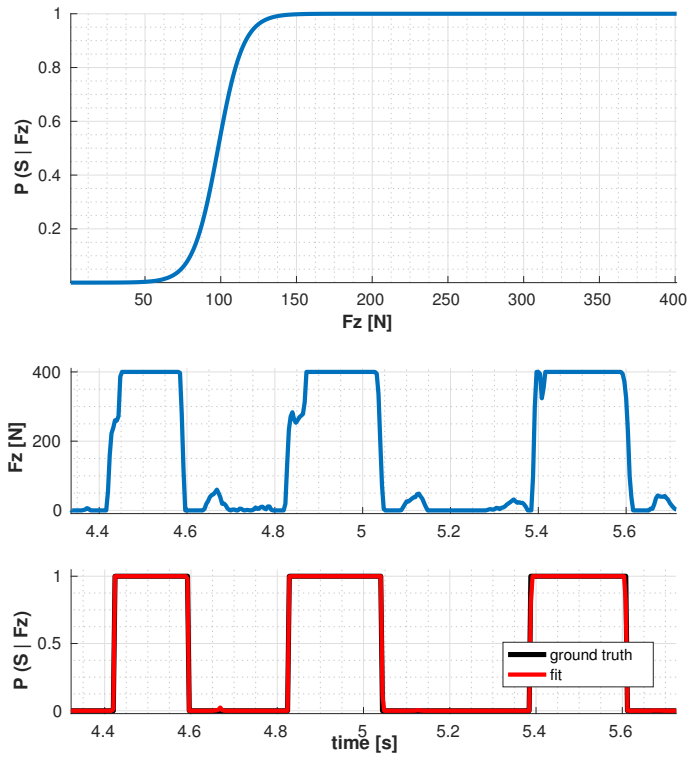


Figure 4.2: Trot gait simulation. *Top plot:* learned logistic model function. *Middle plot:* normal component of the GRF for one leg. *Bottom plot:* learned stance probability and ground truth.

Fitting with Real Data

To test the classifier in a real scenario, we performed training on half of a trot log and half of a crawl log from our dataset (see Appendix A.2) and we used the rest of the dataset as a test set for the learned model.

As no suitable commercial solution for contact sensing was available on our hardware, the ground truth for the training was generated with the optimization described in Section 4.2.3. This approach has the advantage that the classifier tends to learn the force threshold beyond which the associated velocity measurement produced by the foot in question becomes reliable.

Figure 4.3 and 4.4 display, for crawl and trot respectively, the obtained logistic function (top plot), the GRF signal (middle plot), and the fitting of the model against the ground truth for the test set (bottom plot). As in the simulation, the threshold for contact activation in the trot gait is higher, with a larger gap between the two gaits in the data collected from real motions.

We compared state estimation performance using our contact estimation approach against two other thresholding methods on $f_{z,l}$: a single threshold method and a Schmitt trigger. Table 4.1 provides an example of how our approach improves the state estimation performance as a function of drift per distance traveled on the x-axis, due to the better selection of the stance legs used for the velocity computation. For these experiments, we decoupled the effect of gyro bias and linear position estimate by using the orientation estimate from a Vicon motion capture

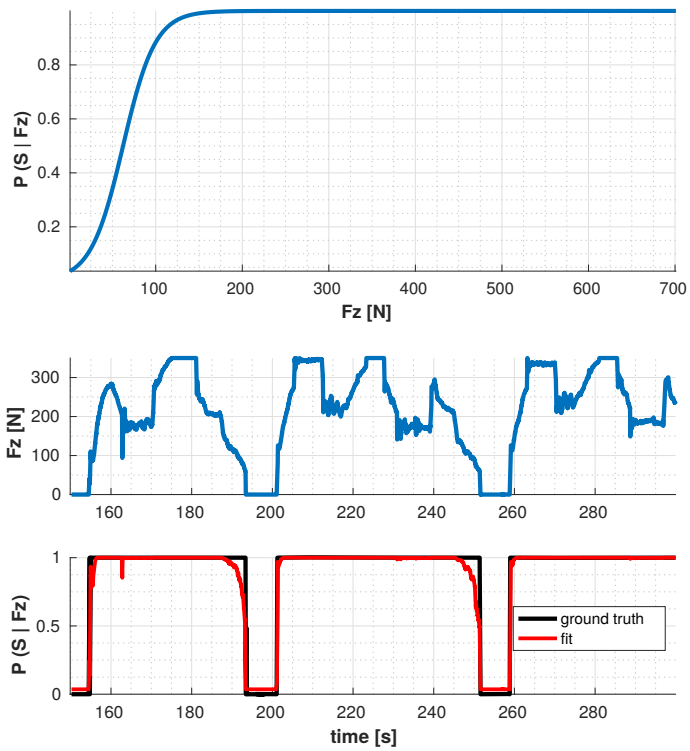


Figure 4.3: Crawl gait experiment. *Top plot:* learned logistic model function. *Middle plot:* normal component of the GRF for one leg. *Bottom plot:* learned stance probability and ground truth.

	Fixed threshold [cm/m]	Hysteresis [cm/m]	Logistic regr. [cm/m]
Crawl	1.34	1.34	1.34
Trot	1.79	0.75	0.43

Table 4.1: DDT of different contact estimators: fixed threshold, Schmitt trigger [104] (hysteresis) and our method (logistic regression).

system. In particular, the proposed logistic regression significantly increases the performance of the LO during the trot gait. For the crawling gait, the performance is equivalent to other methods, since impact events occur less frequently and with reduced intensity.

4.3 Measurement Integration

In the previous section, we described a method to detect which feet are in reliable contact with the ground. This information is essential to form a measurement update for the Kalman update step. In the following, we describe how to compute the mean and covariance for the EKF measurement update, formally described by the measurement function $\boldsymbol{z}_k = h(\boldsymbol{x}_k, \boldsymbol{w}_k)$.

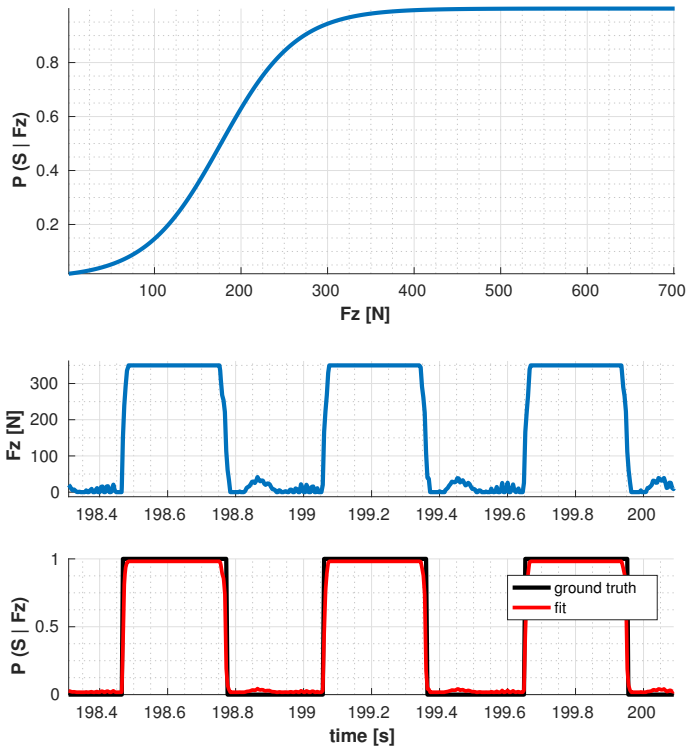


Figure 4.4: Trot gait experiment. *Top plot:* learned logistic model function. *Middle plot:* normal component of the GRF for one leg. *Bottom plot:* learned stance probability and ground truth.

4.3.1 Velocity Estimation

Given an estimate of the feet that are most likely to be in reliable contact, we compute a velocity estimate of the base ${}_b\dot{\mathbf{x}}_b$ and its associated covariance matrix $\Sigma_v = \text{diag}(\sigma_x^2, \sigma_y^2, \sigma_z^2)$ using kinematic sensing. This will then be used as a measurement update in the EKF update step. To compute the estimate we use the contact estimate introduced in Section 4.2.2, while to compute the covariance we leverage the knowledge about the consistency between the velocity contributions of the stance legs and the detection of impacts.

To produce a base velocity update for the filter, we combine the individual base velocity estimates produced by each leg. The most straightforward way to compute this velocity is to compute the simple average of the stance feet:

$$\dot{\mathbf{x}}_k = \mathrm{E}[\dot{\mathbf{x}}_{k,l}] = \frac{1}{\dim(C)} \sum_{l \in C} \dot{\mathbf{x}}_{k,l} \tag{4.17}$$

where C is the set of feet detected as in stance, and $\dim(C)$ is its cardinality.

Since for each velocity contribution we have access to the probability of contact $\mathcal{P}_k$, we can leverage this information to produce a better estimate than the one from Equation 4.17. At each timestep k, we can use the probability of a given foot related to leg l as a weighting criteria for

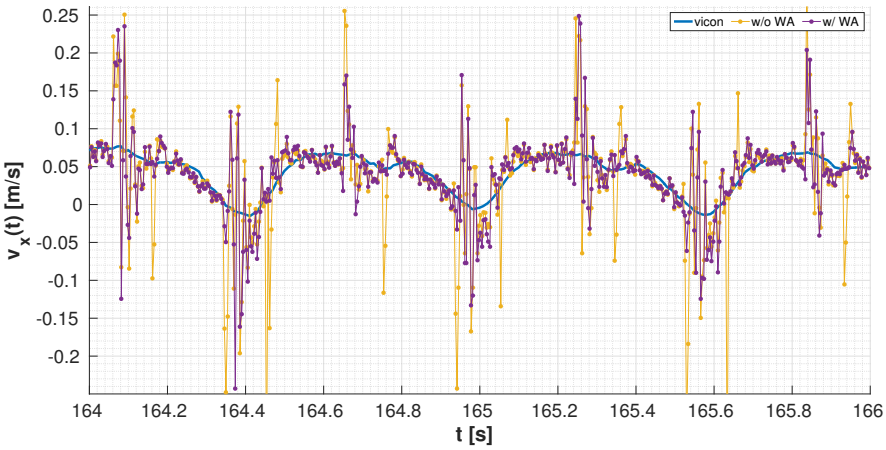


Figure 4.5: Comparison between velocity on x-axis computed using contact estimation with simple average (yellow), weighted average (purple) and ground truth (blue). Note that the weighted average method avoids part of spurious velocity peaks. The data was recorded from a trotting gait experiment (Dataset 2 of Appendix A.2).

the measurement:

$$\dot{\mathbf{x}}_k = \frac{\sum_{i \in C} \mathcal{P}_k(s_l = 1 | \mathbf{f}_l^k) \dot{\mathbf{x}}_{k,l}}{\sum_{i \in C} \mathcal{P}_k(s_l = 1 | \mathbf{f}_l^k)} \tag{4.18}$$

where C is the set of feet that exceed the 0.5 threshold of the logistic regressor. In Equation 4.18 the importance of a velocity contribution from a leg is proportional to its probability of being in reliable contact. Figure 4.5 shows a comparison between the velocity estimation from Equations 4.17 and 4.18. The weighted average reduces the number of negative peaks (yellow line) by giving more importance to reliable legs. Note that not all the peaks are removed, since impact forces can create unrealistic velocities for all the legs.

4.4 Covariance Estimation

Correctly estimating the covariance of these velocity contributions is particularly important. The robot executes different types of dynamic gaits and creates unrealistic velocity updates when a foot strikes the ground.

4.4.1 Inter-Leg Variance

For simplicity, let us assume that each velocity contribution $\dot{\mathbf{x}}_{k,l}$ comes with the same diagonal covariance matrix, Σ_0:

$$\Sigma_0 = \begin{bmatrix} \sigma_{0,x}^2 & 0 & 0 \\ 0 & \sigma_{0,y}^2 & 0 \\ 0 & 0 & \sigma_{0,z}^2 \end{bmatrix} = \Lambda(\sigma_{0,x}^2, \sigma_{0,y}^2, \sigma_{0,z}^2) = \Lambda(\sigma_{0,x}, \sigma_{0,y}, \sigma_{0,z})^2 \tag{4.19}$$

Without further assumptions, the mean of the Gaussian distribution over the measurement $\bar{\boldsymbol{\mu}} = \dot{\mathbf{x}}_k$ is given by Equation 4.17. Similarly, the covariance matrix Σ_v is given by:

$$\begin{aligned}\Sigma_v &= \mathrm{Var}[\dot{\mathbf{x}}_k] \qquad (4.20)\\ &= \mathrm{E}\left[\,\mathrm{Var}[\dot{\mathbf{x}}_k|\dot{\mathbf{x}}_{k,l}]\right] + \mathrm{Var}\left[\,\mathrm{E}[\dot{\mathbf{x}}_k|\dot{\mathbf{x}}_{k,l}]\right]\\ &= \Sigma_0 + \frac{1}{\dim(C)}\sum_{l\in C}(\boldsymbol{\mu}_l - \bar{\boldsymbol{\mu}})(\boldsymbol{\mu}_l - \bar{\boldsymbol{\mu}})^{\mathrm{T}}\\ &= \Sigma_0 + \frac{1}{\dim(C)}\sum_{l\in C}(\dot{\mathbf{x}}_{k,l} - \dot{\mathbf{x}}_k)(\dot{\mathbf{x}}_{k,l} - \dot{\mathbf{x}}_k)^{\mathrm{T}} \qquad (4.21)\end{aligned}$$

Equation 4.21 states that the covariance of the final measurement, composed by the fusion of multiple measurements (each one coming from stance legs $l \in C$), is the mean of the covariances of the single leg distributions Σ_0, plus the covariance of the means of those distributions. This result reflects the fact that legs deemed to be in stable contact with the ground should provide the same estimates for the same base velocity. Ideally, if all the contributions had the same value, the second term of Equation 4.21 would disappear and Σ_v would be reduced to Σ_0. In contrast, high inter-leg covariance would increase the uncertainty of the distribution on the estimated base velocity.

4.4.2 Impact Detection

When the foot strikes the ground, the effect of impulsive forces induce a contact, since the GRF has a high norm. However, the corresponding velocity contributions initially produce unrealistic measurements which propagate to all the legs, as shown in Figure 4.6. During these events, the uncertainty of these measurements increases dramatically. Impact events are characterized by discontinuities in the GRF. Given the normal component of the GRF at time k and $k-1$, we can compute its absolute difference as $|f_{z,l}^k - f_{z,l}^{k-1}|$. If we compute the average value over the stance legs, we can define the following quantity:

$$\Delta_{k,f} = |\Delta \bar{f}_z^k| = \frac{1}{\dim(C)}\sum_{l\in C}|f_{z,l}^k - f_{z,l}^{k-1}| \qquad (4.22)$$

Equation 4.22 defines a metric to measure the intensity of an impact force. It is calculated as the absolute value of the average difference of the normal component of the GRF. Now, we want to incorporate this information into the covariance matrix of Equation 4.21. The second term of Equation 4.21 is a sum of squared matrices. For simplicity, we reject the cross-correlation terms and keep only the diagonal terms of this sum:

$$\begin{aligned}\Sigma_v &= \Sigma_0 + \frac{1}{\dim(C)}\sum_{l\in C}(\dot{\mathbf{x}}_{k,l} - \dot{\mathbf{x}}_k)(\dot{\mathbf{x}}_{k,l} - \dot{\mathbf{x}}_k)^{\mathrm{T}}\\ &\approx \Sigma_0 + \Lambda(\sigma_{e,x}, \sigma_{e,y}, \sigma_{e,z})^2 \qquad (4.23)\end{aligned}$$

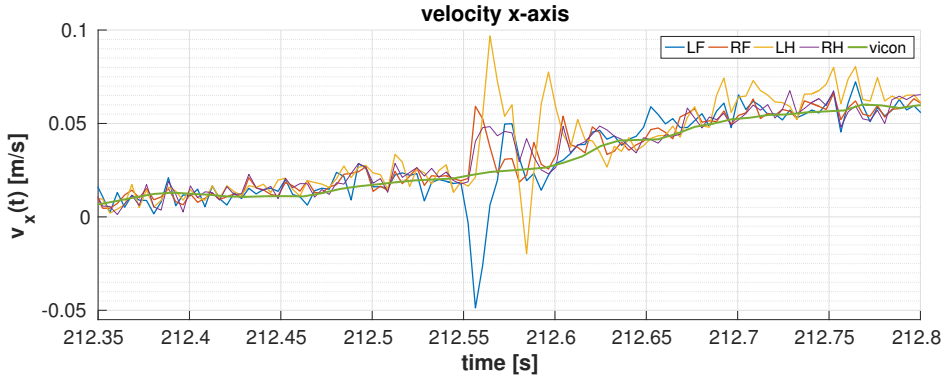


Figure 4.6: Effect of impulsive force on estimated velocities during a crawl gait. The Left Hind (LH) leg strikes the ground at time $212.55\,\mathrm{s}$ producing unrealistic velocity estimates for that leg — as well as for the other legs, due to propagation of the impact on the rest of the structure and concurrent redistribution of the robot's weight over the support legs.

When an impact occurs, the uncertainty increases proportionally to $\Delta_{k,f}$. We incorporate this fact in Equation 4.23 by doing a linear combination of the two effects of inter-leg covariance and impacts:

$$\Sigma_v = \Sigma_0 + \left[\frac{1}{2}\Lambda(\sigma_{e,x}, \sigma_{e,y}, \sigma_{e,z}) + I_3 \frac{\Delta_{k,f}}{2\alpha}\right]^2 \tag{4.24}$$

where α is a normalization factor, computed as the ratio between typical velocity error against the ground truth and $\Delta_{k,f}$ at the same instant.

In Figure 4.7, we show an example of the adaptive covariance described in Equation 4.24, on data extracted from a trot log. We compare the raw (*i.e.,* not yet processed by the EKF) base velocity computed from Equation 4.18 and the ground truth, on the x-axis. The velocity is colored proportionally to the standard deviation on the x-axis extracted from Equation 4.24. Note the change of color in the proximity of feet contact transitions and impacts, where the standard deviation is increased from $0.03\,\mathrm{m/s}$ to $0.13\,\mathrm{m/s}$. During these intervals, confidence in the velocity updates processed by the EKF is reduced.

4.5 Experimental Results

In this section, we describe the performance assessment of the filter presented above. A series of experiments, which include the execution of both dynamic and quasi-static gaits for over an hour, have been carried out to validate the robustness of the estimator.

4.5.1 System Overview

Figure 4.8 shows an overview of the EKF-base framework used on HyQ: acceleration and angular velocity are measured directly from the IMU at $500\,\mathrm{Hz}$ via USB and synchronized with the other

Figure 4.7: Raw velocity on x-axis compared to ground truth during a trot motion. The standard deviation associated to the velocity samples is shown with a color scale, ranging from dark blue ($0.02\,\mathrm{m/s}$ to dark red $0.15\,\mathrm{m/s}$).

signals passively [102]. Joint position, velocity and effort are provided by a real-time process which synchronously samples the encoders and joint force/torque sensors at $1\,\mathrm{kHz}$. These signals are then transmitted to the filter at $250\,\mathrm{Hz}$. Thus, on average, the inertial prediction is executed twice before a measurement is available.

The three blocks in green, which are part of the same UNIX process, are: IMU (Section 4.1.1), Stance Detection (implementation of the stance detector explained in Section 4.2) and Leg Odometry, explained in Section 4.3.1. These constitute the sources required to compute the prior and the posterior distributions for the filter. The filter output is used by the controller directly (red block). The filter is modular and accepts multiple occasional or low-frequency position and velocity estimates (blue blocks at the bottom), as we will see in Chapter 5.

4.5.2 Performance Evaluation

Figure 4.10 compares, for a forward trot, the velocity estimates before filtering (top plot), after filtering (middle plot) and the ground truth (bottom plot). Despite several spikes due to impacts, the filtered output is remarkably smooth. This is thanks to the adaptive covariance algorithm presented in Section 4.3.1, which automatically reduces the confidence in the kinematics filter updates during stance transitions.

In Figures 4.11 and 4.12, we show the average performance on the dataset presented in Appendix A.2, with a distinction between trot and crawl logs, as well as between coordinates. In Figure 4.11 we evaluate the DDT (*i.e.,* the mean position drift divided by the total distance covered by the robot, see Appendix C), while Figure 4.12 shows the Root Mean Square Error (RMSE) of the velocity estimates. For comparison with the proposed algorithm (yellow bars), we provide a simple method (dark blue bars) based on a fixed threshold of $50\,\mathrm{N}$ on the normal component of the GRF for contact detection, and static covariance for the velocity updates.

Although the two gaits we used for our tests differ considerably, for both we noticed a

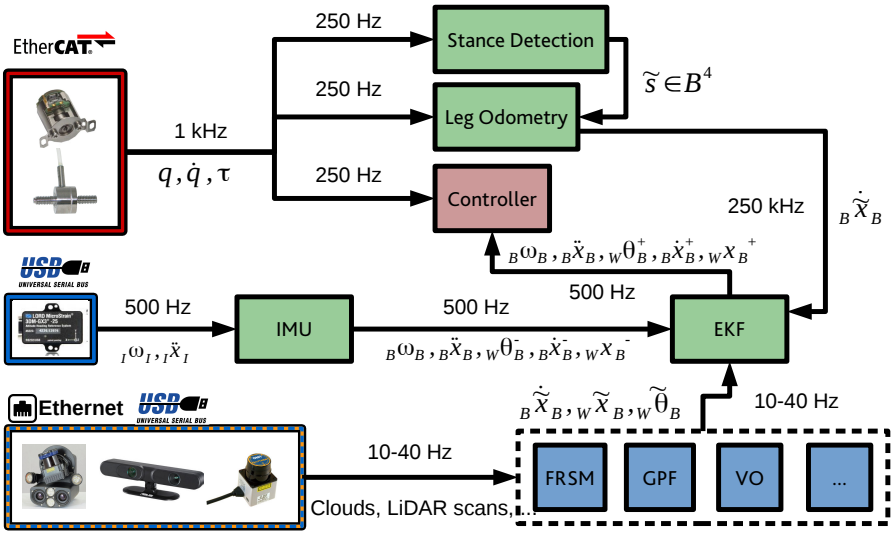


Figure 4.8: Schematic of the EKF framework used on HyQ.

performance degradation on the y-axis – an issue we attribute to structural flexibility of the leg. We provide more detailed discussion about this issue in Section 4.6.

Trot logs

For the trotting logs (left-hand sides of Figures 4.11 and 4.12), we demonstrate that properly handling of impacts significantly improves the performance in both velocity and position. We have achieved this in the x- and the z- axes, where the error in position is more than halved with respect to the simple method (dark blue bars in Figure 4.11. See also Figure4.9 for compared trajectories). In the y-axis the same performance improvement was not obtained. We discuss the issue of limb flexibility in that axis in Section 4.6.

Crawl logs

The plots on the right-hand side of Figures 4.11 and 4.12 show the two main performance indicators for the crawling logs. As expected, given the sporadic occurrence of impacts, the improvement provided by our proposed approach was limited. We notice how the error on the z- component is lower than for the trot because of the continuous support typical for this gait.

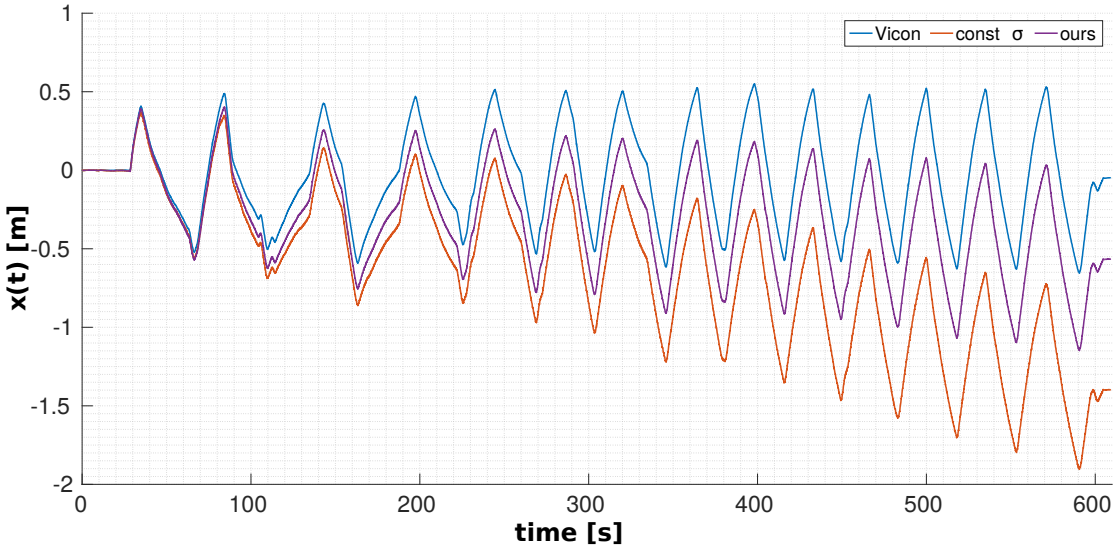


Figure 4.9: Position comparison (on the x-axis) between baseline method (red) and our proposed method (purple). The ground truth is provided by a motion capture system (blue).

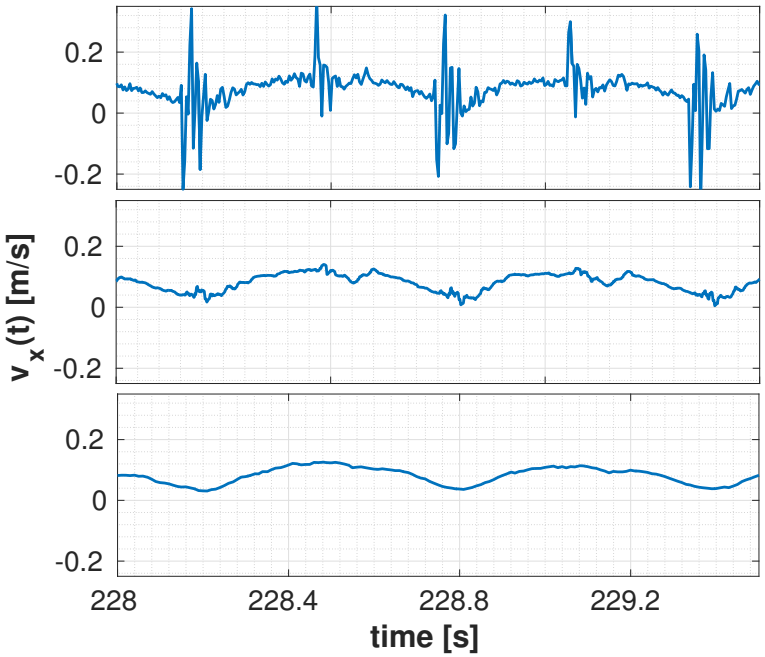


Figure 4.10: Example of velocity estimation for a trot log on x-axis. *Top plot:* raw velocity. *Middle plot:* output of the EKF. *Bottom plot:* ground truth.

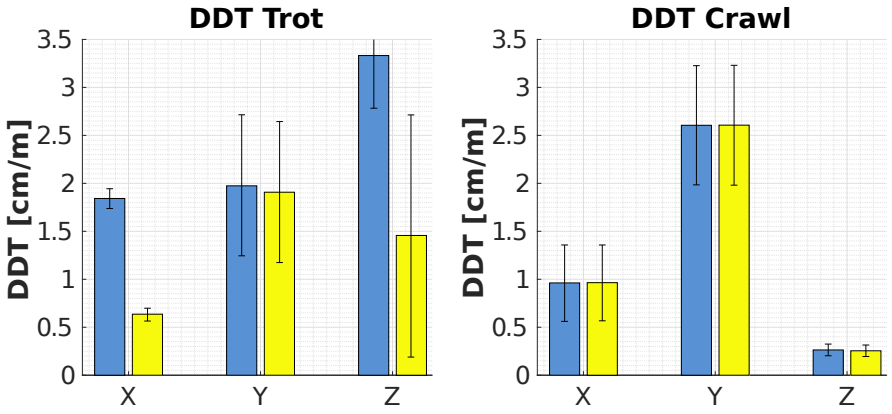


Figure 4.11: Drift per Distance Traveled (DDT) for trot and crawl logs. Left light blue bars show the baseline method, right yellow bars show our approach. The whiskers indicate ± 1 standard deviation computed over the dataset used.

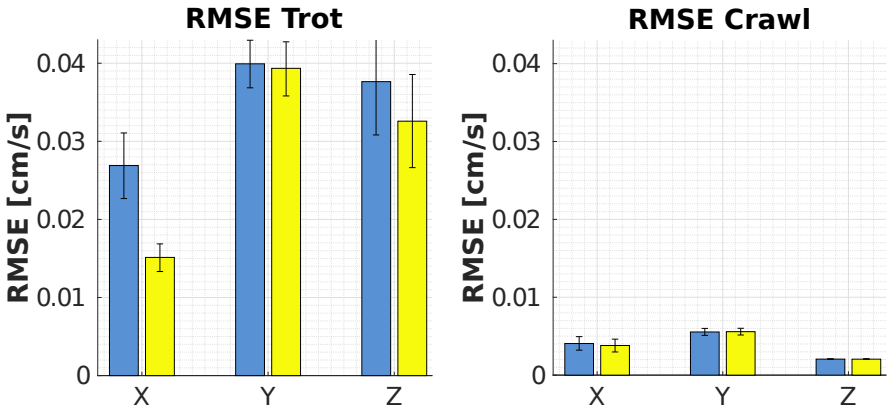


Figure 4.12: RMSE of the velocity estimates for trot and crawl logs. Left light blue bars show the baseline method, right yellow bars show our approach. The whiskers indicate ± 1 standard deviation computed over the dataset used.

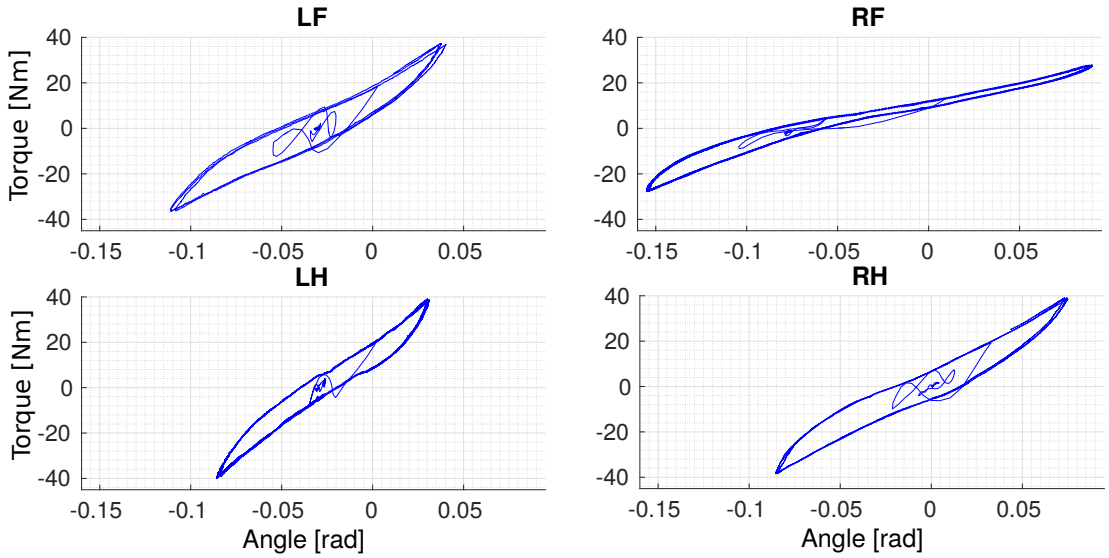


Figure 4.13: HAA Torque vs. Position curves of the four legs under loading/unloading produced by a triangular wave with period 20 s and amplitude 70 N at the foot on the y-axis. Note the hysteresis due to damping between loading and unloading.

4.6 Discussion

In this section, we discuss the limitation of kinematics-inertial state estimation. We focus in particular on the mechanical structure and the proposed training approach.

4.6.1 Leg Compliance

Large robots like HyQ exhibit leg compliance and flexibility when their feet strike the ground, even at slow speeds. Typical forces at the feet are in the range of 200–600 N while crawling or trotting, and beyond 1200 N while bounding. These forces are partially absorbed by the leg structure.

A dedicated experiment shows that the major cause of performance degradation along the y-axis (shown in Figures 4.11 and 4.12) is the intrinsic flexibility of the legs on the coronal plane. With the robot base fixed in place and the feet firmly in contact with the ground, we controlled the robot to produce lateral forces at the foot along the y-axis, using a triangular wave with period of 20 s and intensity of 70 N. Figure 4.13 shows the relationship between position and applied torque at HAA joints (see Chapter 3) which highlights the nonlinearity of the leg structure and the hysteresis between loading and unloading phases. Given the configuration of the experiment, the joint motion should have been very small, while instead a range of 0.24 rad is recorded (see the widths of the graphs in Figure 4.13). This indicates significant structural flexibility.

The development of methods to model this nonlinearity, in order to achieve the same performance on the y-axis as was obtained on the x-axis, is a subject of ongoing research. We are also interested in testing the approach on the second version of HyQ, called HyQ2Max [130],

which is expected to show a better structural behavior.

4.6.2 Limitations

Besides the flexibility in the mechanical structure of the robot, limitations lie in a) the training of the contact classifier and 2) terrain properties. For the whole dataset, only two training phases were sufficient, one for each gait, but new trainings are needed for every new gait or loading condition for the legs. This could be avoided by performing unsupervised learning and active exploration of terrain frictional properties, but at the current stage, only tests in controlled single leg setups have been reported in literature [59]. An alternative solution could be simulation, which would provide a set of parameters for a sufficient number of cases in order to generalize the applicability of the approach.

Concerning the terrain properties, in order to correctly estimate contacts, GRFs need to be projected on the local plane where the foot is experiencing the contact. Although this can be done in first approximation by fitting a plane through the current or recent stance feet positions, more sophisticated methods (using exteroception) are required when the terrain inclination changes significantly within the support region. Other terrain properties, like elasticity or plasticity, are not explicitly accounted for, but a contact model for specific terrain classes can be learned through the proposed approach.

4.7 Conclusion

In this chapter, we presented a state estimation filter which uses inertial measurements and kinematics to estimate position and velocity of the robot's base. The filter, originally designed by Bry *et al.* [22], fuses two sources: an inertial process model and a novel LO module. In contrast to previous works, which rely on foot sensors, our novel LO module uses a probabilistic approach to contact estimation, which makes it suitable for dynamic legged robots without contact sensors. The LO algorithm uses a logistic classifier to learn the GRF threshold with the highest probability to minimize the velocity error against the ground truth. Additionally, it probabilistically merges the velocity contributions from the individual legs in order to create the main measurement update. Furthermore, instead of using fixed covariance, the module dynamically computes the uncertainty associated to the measurement, accommodating for the typical foot-ground impacts of a dynamic legged machine. We have demonstrated that the combination of these two new algorithms (contact estimation and covariance estimation) can double the performance in estimated position and velocity (when compared to standard methods) and that it can compensate for the lack of dedicated contact sensors at the feet. The method have been extensively tested for more than one hour on a quadrupedal robotic platform without contact sensors, in both quasi-static and dynamic locomotion regimes. This work has been published on [27] as first author.

In the next chapter, we will focus on the integration of exteroceptive pose estimation methods, to eliminate the drift on the non-observable states (*i.e.,* linear position and yaw).

Chapter 5

Multisensor Fusion for Accurate Pose Estimation

In the previous chapter, we introduced an EKF which fuses proprioceptive sensing (kinematic and inertial) for state estimation of the dynamic quadruped robot HyQ. Proprioceptive sensing provides an essential basis for legged state estimation: it comes at a high rate ($> 100\,\mathrm{Hz}$), which is critical to control the robot, and it does not depend on environmental conditions (*e.g.,* lighting, textures, *etc.*). However, without exteroceptive inputs, the linear positions and yaw orientation are not observable [17, 16]. Hence, the drift over these states increases over time. In this chapter, we analyze different algorithms to limit and eventually eliminate the drift (by fusing together proprioceptive and exteroceptive sources) and to achieve a smooth and accurate pose estimate.

Accurate pose estimation has been explored and tested extensively for wheeled and flying robotic platforms, while fewer examples of real scenario applications on dynamic legged robots exist. The main differences between estimating robot poses of wheeled or flying platforms rather than legged platforms are: the average speed, the smoothness of the trajectory, and the accuracy requirements.

In typical operative conditions, wheeled and flying robots are generally faster than legged machines, but their trajectory is expected to be smoother. The advantage of having intermittent contact with the ground, which makes legged robots naturally superior in terms of rough terrain traversability, comes at the price of harsher motions [12]. This is challenging for vision based localization, because the repetitive camera shaking causes motion blur, and it breaks the assumption of small incremental motions in between camera frame acquisitions.

In general, wheeled and flying mobile robots have less restrictive requirements in terms of pose estimation accuracy. If we exclude extremely aggressive maneuvers, for which dedicated hardware and software is being developed [79], a few cm of pose uncertainty can be tolerated with no harm for a flying robot, but it is critical for a legged robot which has to select a safe sequence of footholds on a dangerous terrain.

Finally, cars and drones can exploit some characteristics of their trajectories. Most wheeled robots can assume a planar trajectory, which means estimating three DoFs instead of six. On

the other hand, flying robots can take advantage of a bird's-eye view, which is useful for feature tracking and uniform sensor sampling [134]. In contrast, none of these characteristics can be exploited by legged machines.

For these reasons, accurate pose estimation of dynamic legged robots remains a challenging and hot topic of research. These platforms have significantly different requirements and operate in different conditions from their other mobile counterparts. Therefore, the extension of state-of-the-art algorithms (designed for wheeled or flying robots) to legged platforms is an attractive subject for research.

After describing the requirements of a dynamic legged robot, in this chapter we will explore a number of sensor fusion solutions to achieve accurate base pose estimates.

The first solution is a novel pose estimation method, which combines inertial measurements and point cloud registration. We introduce a new ICP variant suitable for real-time computation on small depth sensor devices. The initial guess for the registration is computed from inertial measurements. This algorithm, which was developed independently from the EKF framework introduced in the last chapter, was the first stand-alone solution for real-time robot localization implemented on HyQ.

The other three pose estimation methods presented throughout the chapter are incorporated into the EKF of Chapter 4, as shown by the blue boxes in Figure 4.8. The first integrated solution incorporates a bidimensional LiDAR registration algorithm into the estimation process, the Fast and Robust Scan Matcher (FRSM) [5]. This state-of-the-art localization method was originally designed for drones moving horizontally. With a series of experiments on our datasets, we show that the drift is almost negligible for horizontal motions. The second sensor fusion solution integrates a 3D localization algorithm, also designed for drones [22]. Based on GPF, this method allows 3D localization when accurate prior maps are available. In this case, the integration of LiDAR information shows good results on HyQ. However, when progressively moving towards real scenarios, we need to integrate more than one exteroceptive sensor modality, in order to overcome the complexity due to the many conditions in which a legged robot is expected to operate. The last sensor fusion solution we present is a novel pose estimation method, which gracefully combines the strengths of four different sensor modalities: inertial, kinematics [27], stereo VO [64], and LiDAR registration [101]. We show that this "hierarchy" of modules provides a smooth, stable and accurate pose estimate, which can be safely used within the control loop for precise autonomous navigation.

The remainder of the chapter is structured as follows: Section 5.1 states the requirements for pose estimation on dynamic legged robots; Section 5.2 describes a novel ICP variant for localization on depth sensor with small FoV; in Section 5.3 we briefly describe the LiDAR registration method of [5]. Then, we provide a performance study of its integration in our framework and robotic platform; similarly, in Section 5.4, we provide a description and performance study of a GPF based localization method [22], also running on HyQ; Section 5.5 shows our most complete approach to sensor fusion for accurate pose estimation. The method has been tested in long duration experiments in a challenging test area – the chapter is concluded by Section 5.6

where we discuss the performance and prospects of the presented approaches, and draw final conclusions.

The material presented in Section 5.2 has been published in [26] as first author. The material from Section 5.5 is currently under review (at the time of writing this dissertation).

5.1 Requirements

The ultimate goal of the HyQ project is to achieve versatile, semi-autonomous locomotion in open spaces as well as constrained environments, such as partially collapsed buildings with debris and other hazardous obstacles along the way (*e.g.,* low or overhanging pipes, staircases, stepping stones, holes). The size of its feet, approximately $4\,\mathrm{cm}$ in diameter, poses a very stringent requirement in terms of localization accuracy, with a desired target from $5\,\mathrm{cm/m}$ to $1\,\mathrm{cm/m}$ at best. This requirement is motivated by safe execution of footstep planning, posture control, and ultimately autonomous navigation. Additional constraints are imposed by operative conditions, such as: traversal of featureless areas, scarce illumination, limited power consumption and onboard computation. To date, the closest example of a pose estimator developed under these requirements has been reported in [83]. However, the system described therein was not designed for closed loop execution nor to be used in constrained environments, where small scale accuracy (*e.g.,* for foothold planning) is more critical than large scale accuracy (*e.g.,* for global path planning).

5.2 Selective Iterative Closest Point

The ICP is one of the best known methods for point cloud registration [96, 18]. This method has been widely used for offline scene reconstruction as well as online robot localization [108]. However, its performance decreases dramatically if the number of features is small and not uniformly distributed [120]. A typical example is given by a robot moving on a flat surface: since the motion is almost parallel to the floor, the algorithm tends to stop immediately at a local minimum, because it cannot detect the relative motion between two overlapping and sliding planes [29].

The recent introduction of cheap and accurate depth sensors on the mass market has moved forward the research of registration-based localization, including ICP-based. However, when requirements impose limited computing resources, processing a full point cloud ($\approx 300\,\mathrm{k}$ points) at the nominal framerate ($30\,\mathrm{Hz}$) remains difficult. Dropping frames, so as to reduce the framerate, is not a favorable option because larger motions has to be estimated in between frames, which can lead to poor performance. To solve these problems, a desired feature would be the ability to select the smallest subset of points which carries most of the information about the robot's ego-motion between two frames. These points should also incorporate robust geometrical features, in order to avoid ambiguities.

In the computer vision domain, background subtraction techniques proved to be effective for detecting moving objects in the image space [19]. Similarly, good candidates for accurate 3D registration are the points whose pixel intensity change considerably between two consecutive frames in the depth image space. Besides outliers due to noise, these are indeed the points that are likely to carry most of the information about both geometry and motion. In this section, we describe how to extract these points and use them to improve both accuracy and speed of the ICP registration algorithm. The registration is further improved by integrating inertial data for real-time robot localization and mapping.

Note that alternative methods based on single frame image processing would not achieve the same result. For instance, edge detection on depth frames would retain only geometrical information, but not motion. Randomized selection of points would speed up the registration, but with the same or worse accuracy as using the full cloud.

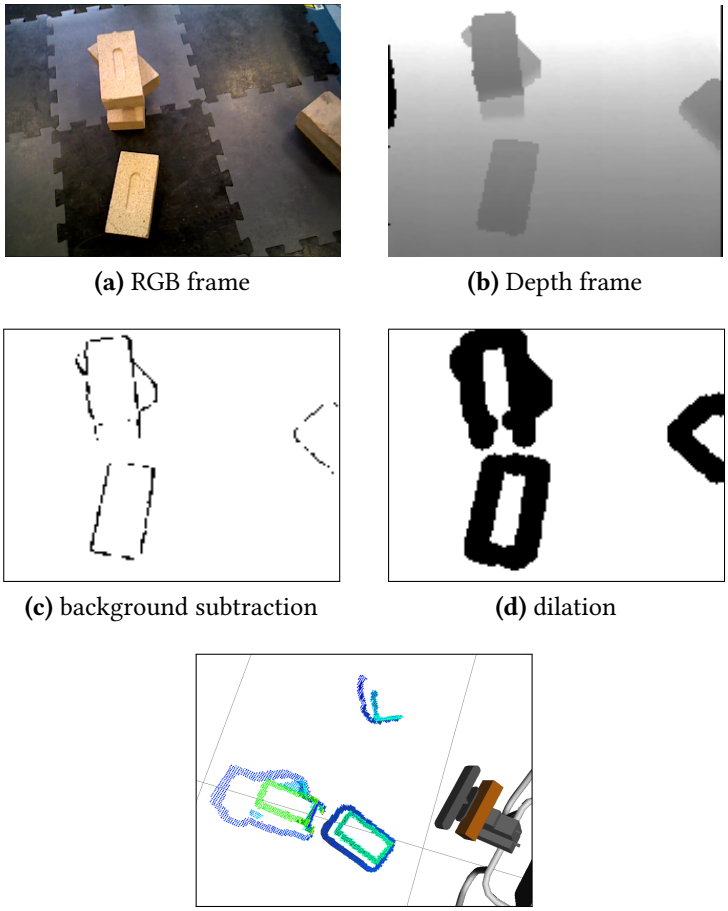
5.2.1 Method Description

We assume, for each timestep k, to have access to the current point cloud $\mathcal{C}_k \in \mathbb{R}^{N\times 3}$ and to have memory of the previous cloud $\mathcal{C}_{k-1} \in \mathbb{R}^{N\times 3}$. Each cloud consists of N tridimensional points $\mathbf{p}_k \in \mathbb{R}^3$ whose coordinates indicate the intersection between a physical surface and the IR light ray projected by the sensor. The clouds are *organized*, *i.e.,* they come arranged into $n \times m$ matrices (with $nm = N$) in accordance with the camera FoV. As a result, an equivalent representation of point clouds is available as a *depth image* $\mathcal{D}_k \in \mathbb{R}^{n\times m}$. The pixel intensities of a depth image, expressed in meters, are proportional to the relative distance between the camera's optical center and the surface being hit by the IR light. Figure 5.1b shows an example of a depth image, converted to grayscale for display purposes. We now describe the processing steps to select the feature point for ICP registration. These are:

Background subtraction: we compute the absolute difference of two consecutive frames $\mathcal{D}_{\Delta,k} = |\mathcal{D}_k - \mathcal{D}_{k-1}|$. Brighter pixels of $\mathcal{D}_{\Delta,k}$ correspond to detectable object motions in the scene. These are moving 3D features (*e.g.,* the bricks' edges in Figure 5.1b), while darker pixels correspond to objects far from the optical center, or portions of the image with apparent absence of motion (*e.g.,* the floor);

Tresholding: the difference image $\mathcal{D}_{\Delta,k}$ is processed by a binary threshold operator. The threshold is manually chosen to filter out most of the background noise (Figure 5.1c). Surprisingly, when using raw depth images (*i.e.,* with floating point values) rather than their grayscale conversion, the background noise was almost absent;

Dilation: for small motions, the thresholded points are sparse and unconnected. We apply a morphological dilation operator to expand the neighborhood of candidate points (Figure 5.1d). The dilation size d is crucial to the performance: very large values are likely to produce big clouds with many featureless areas, while small values can cause instability due to variability between the target and source clouds;



(a) RGB frame **(b)** Depth frame

(c) background subtraction **(d)** dilation

(e) final result

Figure 5.1: Image preprocessing for point selection.

Temporal fusion: to increase the overlapping between the source and target clouds, we perform a bitwise OR of the current dilated frame with a history of previous h frames. Again, the history size h has to be chosen properly to balance speed and accuracy: longer history would increase the number of selected points and decrease the speed;

Mask: the binary image obtained from the previous step is finally used to mask $\mathcal{D}_k$ and $\mathcal{D}_{k-1}$ and obtain the *source* and *target* clouds. An example of the final result is shown in Figure 5.1e. After the processing pipeline, only 10% of the original point clouds have been retained. The brick edges are correctly segmented, while the majority of ground points, which are ambiguous to the matching algorithm, are removed.

To demonstrate the effectiveness of this approach, we run the ICP open-source implementation from [121] on a datalog recorded on HyQ trotting between bricks $1\,\mathrm{m}$ forward and $1\,\mathrm{m}$ backward. Note that the proposed algorithm does not depend on a specific ICP implementation, because it executes a smart selection of the input for the registration, rather than operating at the registration level. Therefore the use of other implementations, such as *libpointmatcher* [109], would have been equally valid.

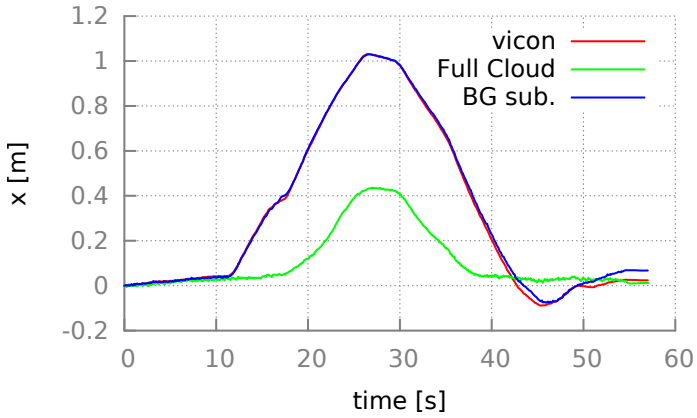


Figure 5.2: Comparison between the estimated robot position on x-axis computed with ICP using the full point cloud (green) and with ICP using background subtraction (blue). The ground truth is provided by a motion capture system (red). Since the majority of the points lies on the same flat surface, with the full cloud the motion is underestimated.

Figure 5.2 shows the estimated x-axis component of the robot's trajectory using the full point cloud (green) and the selected point cloud (blue). The two are compared with the ground truth (red), measured from a Vicon motion capture system. The selection algorithm improves the tracking by more than 100% in the central part of the run. The full cloud, having many more points perceived as static rather than moving 3D features, makes the ICP underestimate the motion. As an additional benefit, the background subtraction reduces the number of points by a factor of 10, allowing it to operate within the real-time constraint. In contrast, with the full cloud, the execution was twice slower than in real-time.

5.2.2 Inertial Measurement Integration

To converge properly, ICP requires a good initial guess for the point cloud registration. In a similar manner to [98], we compute the guess from the IMU. In the following, we assume that point clouds and IMU quantities have been previously transformed into a common reference frame, using the calibration techniques described in Section 3.6.

We use the absolute orientation quaternion $\mathbf{q} \in \mathbb{R}^4$ (estimated onboard the device) and the linear accelerations $\ddot{\mathbf{x}}$ (computed from Equation 4.3). The ICP guess is a relative transformation between two point clouds $\mathcal{C}_k$, and $\mathcal{C}_{k-1}$. Hence, we compute the corresponding angular position difference as the quaternion multiplication $\mathbf{q}_{\Delta t} = \mathbf{q}_k \cdot \mathbf{q}_{k-1}^{-1}$, while the linear position difference is computed from the positions obtained by double integration of the raw accelerations using the trapezoidal rule [111]:

$$\dot{\mathbf{x}}_j = \dot{\mathbf{x}}_{j-1} + \frac{1}{2}(\ddot{\mathbf{x}}_j + \ddot{\mathbf{x}}_{j-1})\,\Delta t_j \tag{5.1}$$

$$\mathbf{x}_j = \mathbf{x}_{j-1} + \dot{\mathbf{x}}_{j-1}\Delta t_j + \frac{1}{2}(\Delta t_j)^2\left(\frac{2\ddot{\mathbf{x}}_{j-1}}{3} + \frac{\ddot{\mathbf{x}}_j}{3}\right) \tag{5.2}$$

Note that the duration of $\Delta t_j = t_j - t_{j-1} \approx 1/250\,\text{s}$ in Equations 5.1 and 5.2 is different from the time elapsed between the generation of the two point clouds $\Delta t = t_k - t_{k-1}$, which is $\approx 1/30\,\text{s}$. When a new cloud is available, the corresponding translation $\mathbf{x}_{\Delta t} = \mathbf{x}_k - \mathbf{x}_{k-1}$ is computed from Equation 5.2 at 30 Hz.

Once the guess from the IMU is obtained, the registration algorithm works as follows (see Algorithm 2): first, we compute the transformation between the two clouds $\mathcal{C}_k$ and $\mathcal{C}_{k-1}$ by means of ICP with guess:

$$\mathrm{T}_k^{\text{IMU}} = \left[\begin{array}{c:c} R(\mathbf{q}_{\Delta t}) & \mathbf{x}_{\Delta t} \\ \hdashline \mathbf{0} & 1 \end{array}\right] \tag{5.3}$$

where $R(\cdot)$ is the function which converts a quaternion into a rotation matrix.

If the result of the registration $\mathrm{T}_k^{\text{ICP}}$ is valid, roll and pitch of $\mathrm{T}_k^{\text{ICP}}$ are replaced with the roll and pitch from $\mathrm{T}_k^{\text{IMU}}$, which are more reliable because they have been computed with the gravitational field (lines 11–12). An ICP transformation is considered valid if:

- the ICP fitness score is below a threshold (of $< 1 \times 10^{-4}$);
- the roll and pitch estimated by the ICP fall in an interval around the IMU roll and pitch;
- the translation and rotation are smaller than the maximum threshold estimated empirically.

In case the transformation is not valid, the linear position part of $\mathrm{T}_k^{\text{ICP}}$ is replaced by the latest valid linear position update (lines 8–9), which corresponds to assuming a constant velocity. The algorithm then continues by accumulating the computed incremental transformation (line 13) and replacing the target point cloud with the source point cloud (line 14).

Algorithm 2 Point Cloud Registration

1: $k \leftarrow 1$
2: $\mathrm{T}_{\text{global}} \leftarrow I_4$
3: getData($\mathcal{C}_0\, \mathbf{q}_0, \ddot{\mathbf{x}}_0$)
4: $\mathcal{C}_{\text{target}} \leftarrow \mathcal{C}_0$
5: **while** getData($\mathcal{C}_k\, \mathbf{q}_k, \ddot{\mathbf{x}}_k$) **do**
6: $\quad \mathrm{T}_i^{\text{IMU}} = \text{computeGuess}(\mathbf{q}_k, \mathbf{q}_{k-1}, \ddot{\mathbf{x}}_k, \ddot{\mathbf{x}}_{k-1})$
7: $\quad \mathrm{T}_k^{\text{ICP}} \leftarrow \text{ICP}(\mathcal{C}_{\text{source}}, \mathcal{C}_{\text{target}}, \mathrm{T}_k^{\text{IMU}})$
8: $\quad$ **if** isInvalid($\mathrm{T}_k^{\text{ICP}}$) **then**
9: $\quad\quad \mathrm{T}_k^{\text{ICP}} \leftarrow \mathrm{T}_{k-1}^{\text{ICP}}$
10: $\quad$ **end if**
11: $\quad \text{roll}(\mathrm{T}_k^{\text{ICP}}) \leftarrow \text{roll}(\mathrm{T}_k^{\text{IMU}})$
12: $\quad \text{pitch}(\mathrm{T}_k^{\text{ICP}}) \leftarrow \text{pitch}(\mathrm{T}_k^{\text{IMU}})$
13: $\quad \mathrm{T}_k^{\text{global}} \leftarrow \mathrm{T}_{k-1}^{\text{global}}\, \mathrm{T}_k^{\text{ICP}}$
14: $\quad \mathcal{C}_{\text{target}} \leftarrow \mathcal{C}_{\text{source}}$
15: $\quad k \leftarrow k + 1$
16: **end while**

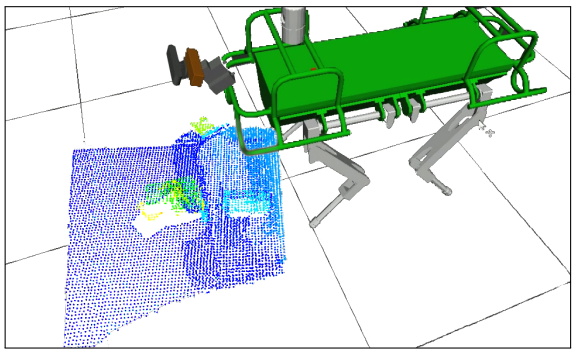


Figure 5.3: Example of online mapping with the Selective ICP algorithm.

5.2.3 Experimental Results

To show the effectiveness of the registration algorithm and its combination with inertial measurements, we performed an indoor trotting sequence with obstacles along the way on our platform HyQ. The ground truth has been measured with a motion capture system. The robot was tele-operated in order to cross different obstacles arranged on a flat surface. Depth images and point clouds were generated at 30 Hz and IMU data at 250 Hz (see Section 3.5.1).

Figure 5.4 shows the estimated position and orientation while the robot was trotting forward and backward in the $2.5 \times 1.2\,\mathrm{m}^2$ test area. We compare the estimated position with the Selective ICP only (green line) and with the Selective ICP fused with IMU measurements (blue line). The ground truth is provided by a Vicon motion capture system (red line).

In general, we can see that both position and rotation estimates of the inertial-registration solution are closer to the ground truth if compared with the ICP-only version, even though the improvement is sometimes marginal ($\approx 1^\circ$ in yaw). As expected, a major improvement is given by the pitch and roll, which are directly taken from the IMU. In both methods, a non-negligible drift in the z-axis is visible. This is due to the lack of features in the zy-plane of the robot, since the camera was facing the ground and therefore the estimated motion was more accurate on the xy-plane.

With our method, the maximum drift after $60\,\mathrm{s}$ of trotting is about $10\,\mathrm{cm}$ on the y-axis, corresponding to 5% of the total path. Note that the robot was aggressively changing direction on the yaw axis, making the estimate of y more difficult than other axes. Nevertheless, the drift was sufficiently low to produce accurate point cloud maps, as depicted in Figure 5.3.

5.2.4 Discussion

The results presented in 5.2.3 highlight the benefits of having a selective algorithm when the features in the scene are outnumbered by geometrically featureless areas. When the robot is moving quickly inside feature-rich areas, the method approximates the standard ICP, since the number of selected points increases and potentially include all of the input data.

The proposed method is general, in the sense that it does not depend on a specific ICP implementation because it operates on the input for the registration rather than the registration

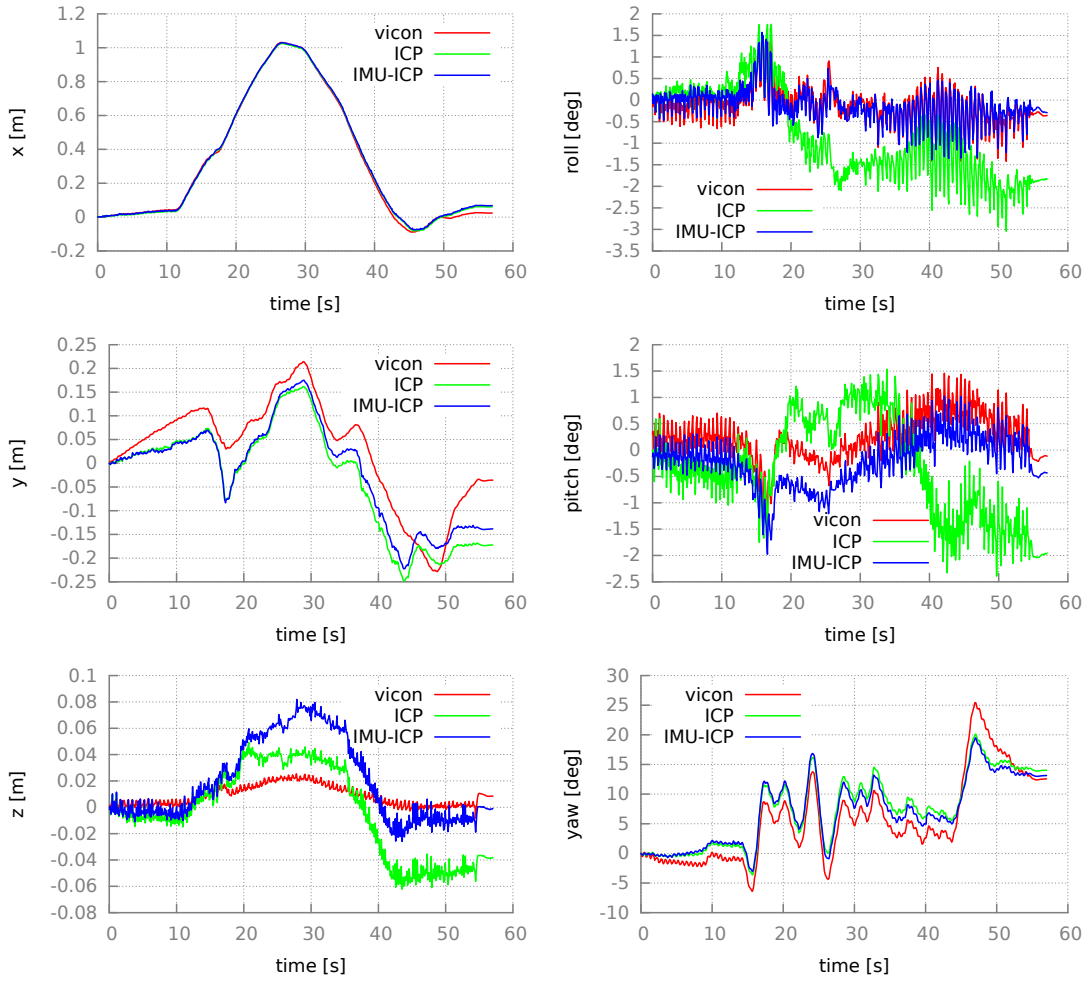


Figure 5.4: Experiment with the real robot trotting indoor. The six graphs represent the 6 DoF pose of the robot in the global frame of reference estimated by the motion capture system (red), the ICP (green) and our framework (blue).

itself. Further optimizations which involve the integration of the IMU (*e.g.,* constraining the ICP search to comply with IMU measurements) are applied indirectly by using it as initial guess for the ICP.

The method does not include mechanisms to detect apparent motion inducted by dynamic environment, which could lead to wrong motion assumptions if the robot is static and a significant portion of the environment is moving in front of it. However, this kind of situation could be handled by fusion with other sources, such as LO.

More sophisticated techniques to handle potential failures due to dynamic scenes, a better integration with the IMU signals, and more extensive tests on different datasets are subject of future work.

5.3 Fast and Robust Scan Matcher

The FRSM is an Inertial-LiDAR based, open-source localization library[1], first introduced in [5] for autonomous quadrotor flight in GPS-denied environments.

Even though quadrotors have 6 DoFs, the library is assuming a multi-level planar trajectory, *i.e.,* triples (x, y, yaw) at different values of z. This allows the decoupling of the pose estimation in two separate problems: planar pose estimation and height estimation. Here we analyze the performance of the former, under the assumption that HyQ will not pitch or roll significantly. This assumption can be considered acceptable only in indoor environments without major unevenness (*e.g.,* obstacles, ramps, staircases). In the following sections, we progressively move towards solutions which can handle more general situations.

The FRSM algorithm of [5] solves the registration problem in the 2D domain with a map-based probabilistic approach rather than frame-to-frame iterative optimization (as ICP does). This guarantees robustness against errors due to slightly non-horizontal motion, as proved by the authors in [5]. In map-based probabilistic scan matching, a grid map M is created from past scans, and incoming scans are matched against that map. The map incorporates, for each cell, the likelihood of a laser return being measured in that location. Assuming that each LiDAR measurement is independent from the other ones composing the scans, the total likelihood for a scan to be measured in a location of the map is the product of the individual likelihoods of each beam composing that scan. The algorithm then performs a search over the rigid body transformations which maximizes the total likelihood for the scan.

In this section, we describe the fusion of the FRSM library with the EKF described in Chapter 4. The output of the library is used as position and yaw updates of the filter, which fuses them together with inertial information and LO. To this end, we mounted a Hokuyo URG-04LX horizontally on the robot's trunk, as described in Section 3.5.2. This sensor is substantially inferior in terms of range ($4\,\mathrm{m}$ vs. $30\,\mathrm{m}$) and update frequency ($10\,\mathrm{Hz}$ vs. $40\,\mathrm{Hz}$) than the one tested on the original library (Hokuyo UTM-30LX).

[1] https://github.com/abachrach/frsm

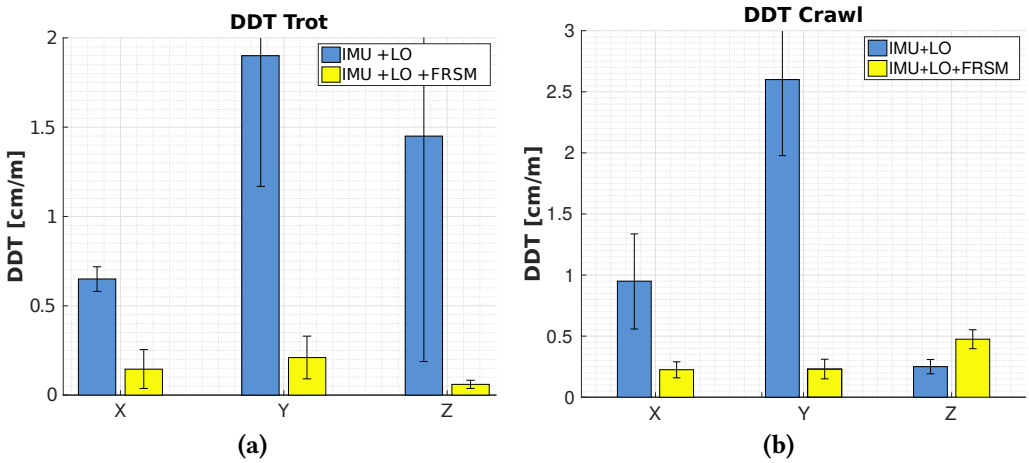


Figure 5.5: DDT on the trot and crawl logs taken from Dataset 2. The light blue bars on the left indicate the estimate fusing IMU and LO, while the yellow bars on the right of each group indicate the estimate fusing IMU, LO and FRSM. The whiskers indicate $\pm$ standard deviation computed over the dataset used.

5.3.1 Experimental Results

In this section, we provide both quantitative and qualitative results of the fusion between IMU, LO and FRSM modules. In particular, we show the pose estimation performance of the filter on Dataset 2 (see Appendix A.2), and we provide an example of application with state estimation used in the control loop.

Pose Estimation Performance

Figure 5.5a and 5.5b compare the performance of the filter in two combinations: using only IMU and LO, and using IMU, LO and FRSM. In almost all cases, the FRSM library is erasing the drift in position and yaw. For the z-axis in the crawl logs, the algorithm is performing slightly worse than the kinematics-inertial combination. This is caused by the different covariance setup for the LO, which affects a state not updated by the scan matcher module. However, the value is within the range of performance requirements (see Section 5.1).

An example of trajectory is shown for a trot log from Dataset 2 in Figure 5.6. We can see that the filter estimate is very close to the ground truth in every condition, except when the sensor is too far or too close to the starting point (bottom peaks on the top plot of Figure 5.6). Note that, in order to achieve reliable performance, the filter has to use all the aforementioned proprioceptive and exteroceptive sources. The IMU and FRSM alone would have provided rougher trajectories, due to the low frequency scan matcher updates (10 Hz). Furthermore, the contribution of kinematics is fundamental for velocity estimation (which is required for in-loop control).

Figure 5.7 shows the orientation performance. Roll and pitch (top and middle plots) are affected by a small bias of 2°. The yaw, instead, has a negligible error, due to the contribution of the FRSM.

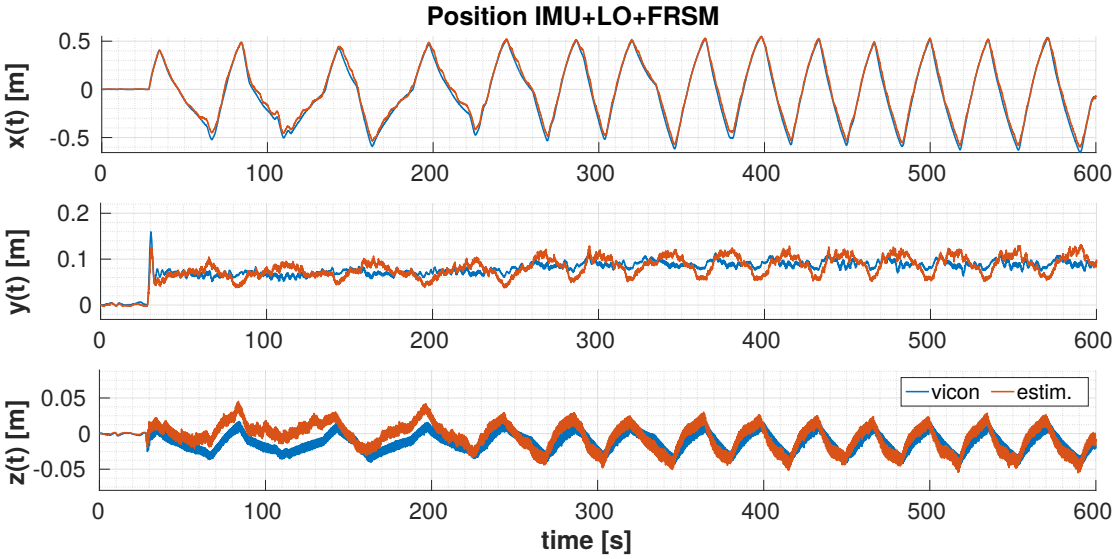


Figure 5.6: Estimated position (*red line*) computed fusing IMU, LO and FRSM modules against Vicon (*blue line*), on a datalog from Dataset 2. The modules are fused with *Pronto* the EKF introduced in [37] and used as framework for the work of [27]

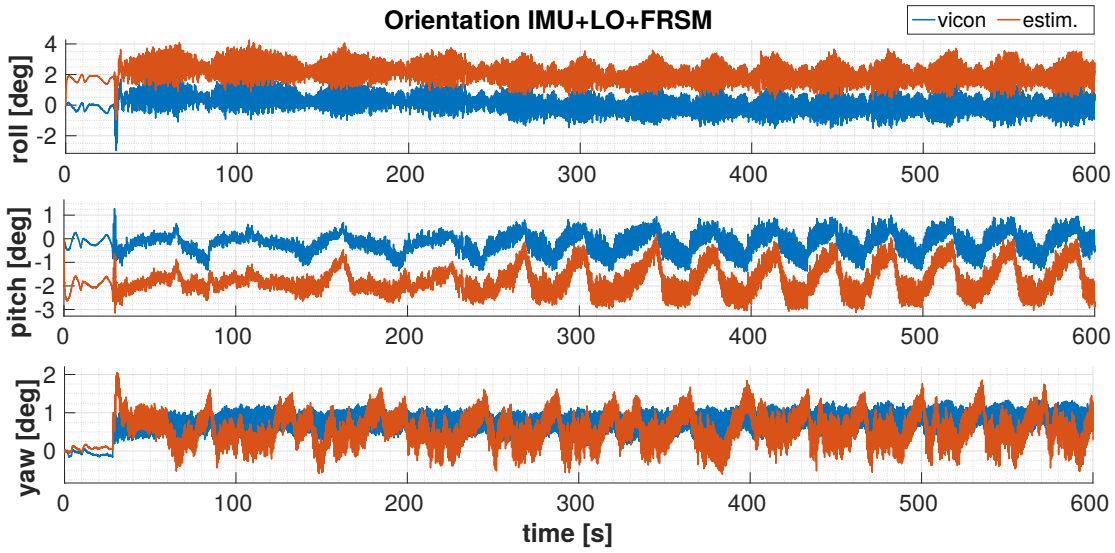


Figure 5.7: Orientation of IMU + LO + FRSM modules (fused with *Pronto*, the EKF introduced in [37] and used in [27]) against Vicon (from Dataset 2).

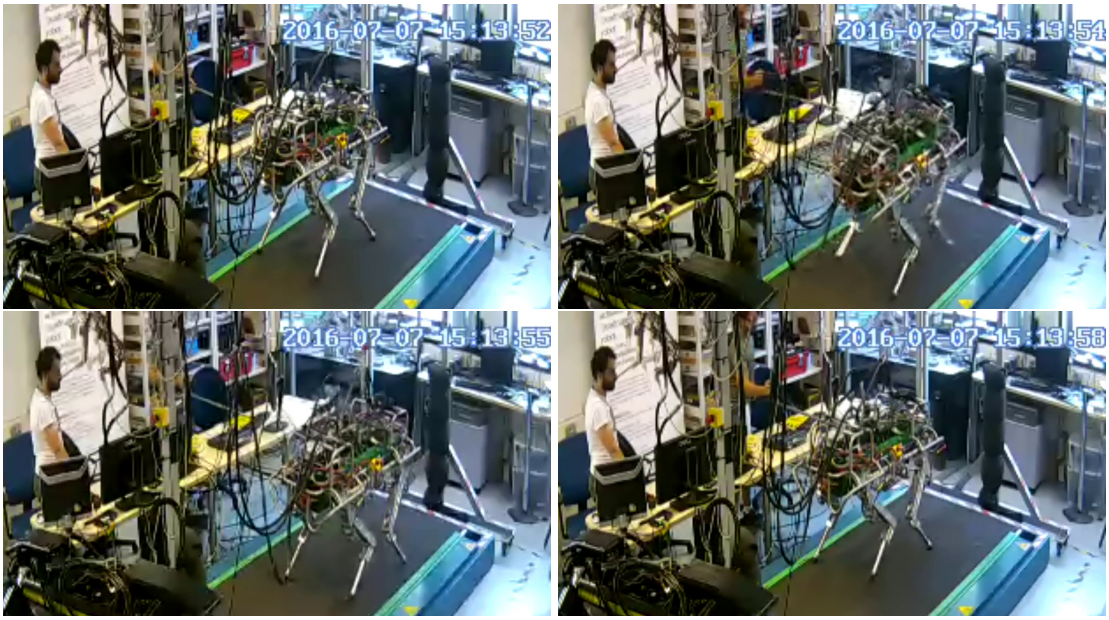


Figure 5.8: Push recovery with state estimation in the loop. The robot was commanded to trot in place and keep its position. After several pushes from the operator, the robot recovered its position along the green line of the treadmill. The ability to recover its position in a constrained space using the state estimation in the loop demonstrates the reliability and robustness of the filter.

In-the-Loop Control Performance

In this section, we describe an example application of the state estimation in the loop for posture and trunk control. Figure 5.8 shows some screenshots from the experiment. The robot is trotting in place and commanded to keep a zero position (green edge beside the treadmill). After a series of disturbances from the operator, the robot returns to the zero position with no drift. The experiment demonstrates qualitatively the robustness and reliability of the estimator. Note that, by the design of the push recovery algorithm [8], both position and velocity estimates are used at the same time. This demonstrates how the filter is gracefully fusing different sensor inputs into a smooth and accurate signal.

5.4 Gaussian Particle Filter

In the previous section, we have shown the performance of the FRSM coupled with inertial and LO by means of an EKF. With the contribution of the scan matcher, the filter produces very accurate pose estimates. However, the scan matcher algorithm is effective only when the robot is moving more or less horizontally. In this section, we relax this constraint with a 6 DoFs localization module based on the GPF algorithm and OctoMap. This module has been originally developed for a fixed-wing Micro Aerial Vehicle (MAV) [22], and it was later extended to humanoids in [37]. In this section, we discuss the performance on HyQ, arranged in Configuration A (see Section 3.5.1). In contrast with the original application, we use a Velodyne HDL-32E, which has a spinning head producing 360° scans composed of 32 rings at $10\,\mathrm{Hz}$ instead of a single linear array at $40\,\mathrm{Hz}$.

5.4.1 Scan Filtering

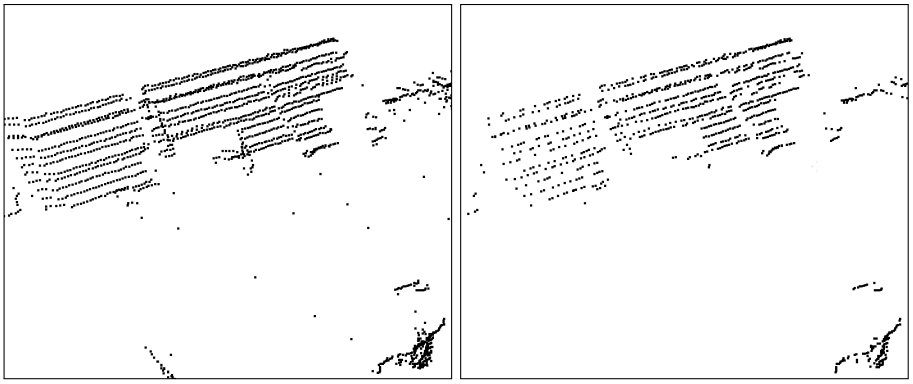
The GPF algorithm of [22] requires a prior 3D OctoMap [61] model of the environment. This is usually built from a sequence of scans fused together when the robot is static or, it is collected manually. In our case, since the point clouds produced by the Velodyne are sparse, we need to run inertial-kinematics state estimation or a motion capture system while the robot is executing push-up motions to fuse multiple clouds from different heights to fill the gaps in the map. During this scan collection process, input filtering is essential. If outlier points are not filtered from the scan while building the model, they become persistent artifacts in the OctoMap, and the localization algorithm performs poorly.

For this reason, we perform intrascan-based filtering before using the scans to build the OctoMap model. We use a differential filter, which discards a point if: a) it is invalid (zero value), b) has an invalid left neighbor, c) the difference between its range value and its left neighbor range value is greater than a threshold. An example of the scan before and after filtering is depicted in Figure 5.9. This filter effectively removes the points in mid-air, which would produce artifacts in the empty space. Given the constrained space and the accuracy requirements, the voxel size of the OctoMap was set to $5\,\mathrm{cm}$.

5.4.2 Experimental Results

In this section, we compare the state estimation performance of: 1) IMU and LO; 2) IMU, LO and FRSM; 3) IMU, LO and GPF. All the tests have been conducted on Dataset 1 (see Appendix A.1).

The summary of the position estimate performance is shown in Figure 5.10. Note that the results for the first two sensor combinations are different from previous sections because the dataset is different (on Dataset 2 the Velodyne scans are unavailable). In general, we see that the FRSM is more effective at reducing the drift. An exception is again given by the z-axis: since the scan matcher is not providing updates on that state, the high covariance set on the LO in order to avoid conflicts with FRSM is causing a decrease in performance. Another example of



(a) Unfiltered Velodyne scan **(b)** Differential filtered Velodyne scan

Figure 5.9: Comparison between unfiltered and filtered point clouds. Note the outlier points in a straight line that are removed after filtering.

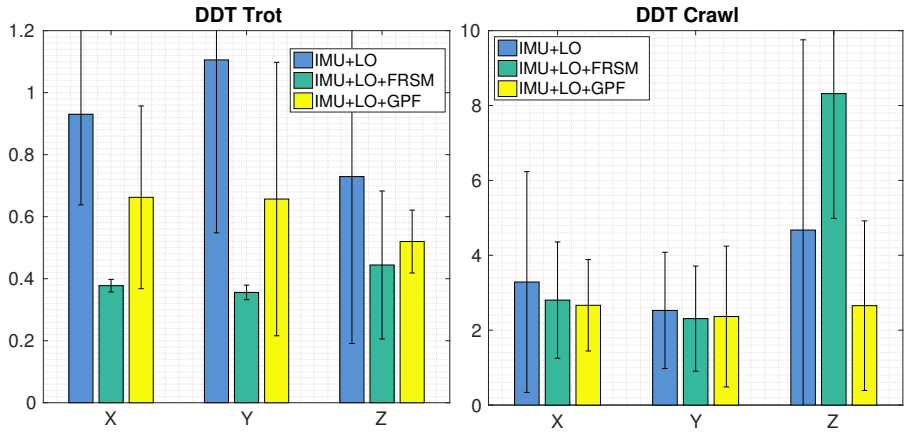


Figure 5.10: Drift per Distance Traveled on trot and crawl logs from Dataset 1.

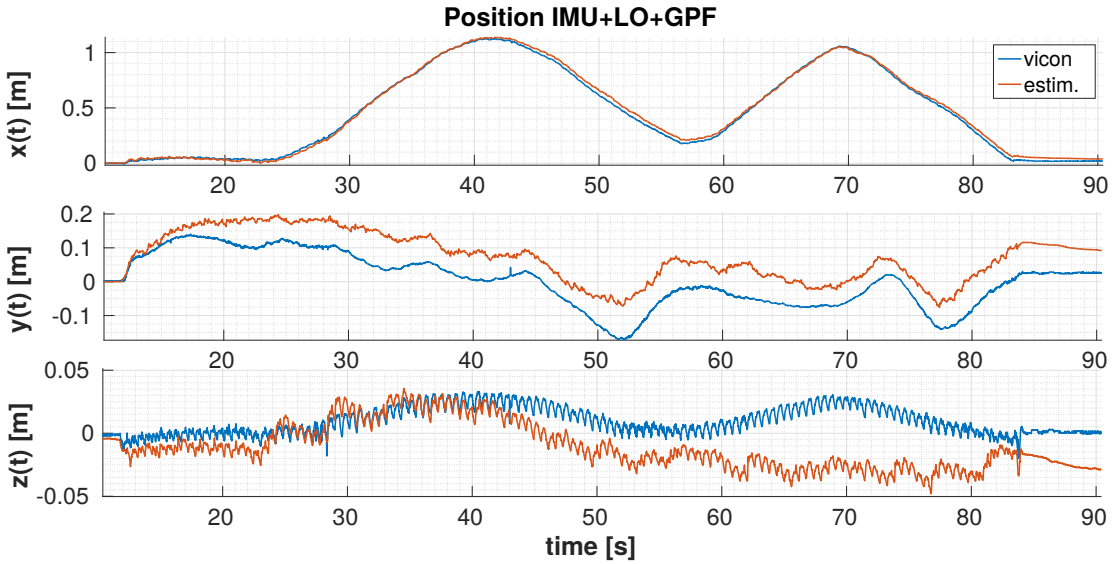


Figure 5.11: Position of IMU + LO + GPF modules (fused with *Pronto*, the EKF introduced in [37] and used in [27]) against Vicon (from Dataset 1).

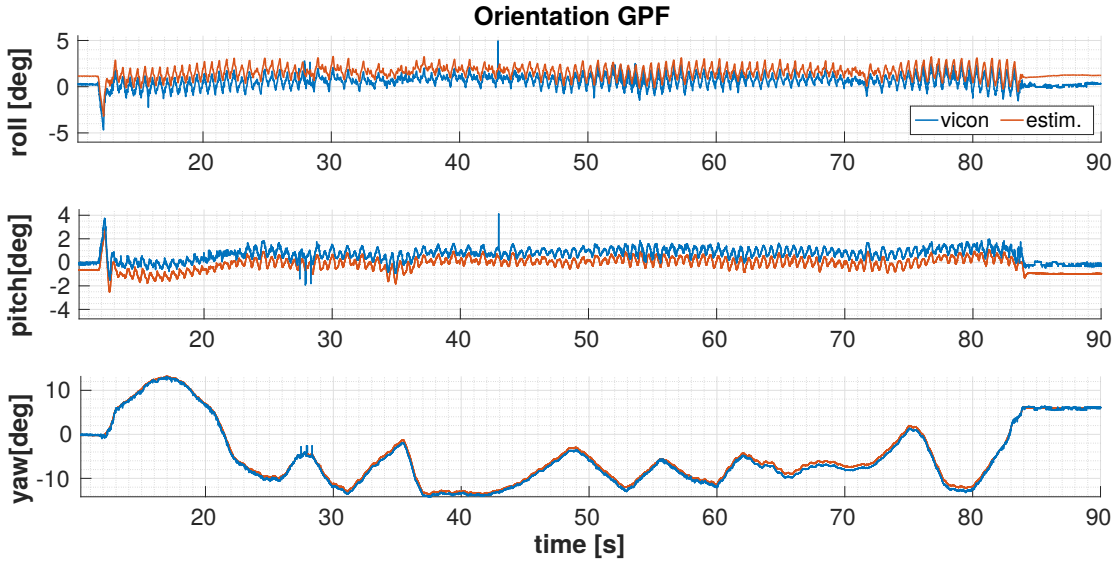


Figure 5.12: Orientation of IMU + LO + GPF modules (fused with *Pronto*, the EKF introduced in [37] and used in [27]) against Vicon (from Dataset 1).

the importance of covariance tuning, to properly balance the fusion of the various signals, is given by Figure 5.11. The known issue on the y-axis from LO (see Section 4.6.1) is propagated to the position estimate and is causing a significant drift ($\sim 10\,\mathrm{cm}$).

5.5 Multisensory State Estimation: Inertial, Kinematics, LiDAR and Visual Odometry

In the previous sections, we have presented three methods to perform accurate pose estimation on the HyQ robot: Selective ICP, Fast and Robust Scan Matcher, and Gaussian Particle Filter. The accuracy obtained by these methods matches the requirements stated in Section 5.1. However, real missions involving HyQ navigating in a disaster scenario impose operative conditions which go beyond the capabilities of these methods. The Selective ICP helps localization on grounds characterized by a lack of 3D features and of depth sensors with very limited FoV. In total absence of 3D features, the restricted FoV would not provide enough clues for ICP to converge. The FRSM is restricted to quasi-planar motions, which rarely occur when navigating outdoors. The GPF appears to be the most general method, since it can operate in a 3D environment. However, the collection of a prior map *in situ* makes its applicability to a real scenario impractical. For example, even if it was working reliably on Atlas [37], it was not used for the Finals of the DARPA Robotics Challenge [101].

Indeed, there is no general localization method and/or sensor modality which can single-handedly generalize well enough to fit the wide spectrum of situations in which a semi-autonomous robot is expected to operate. In this context, sensor fusion of multiple proprioceptive and exteroceptive sources appears to be the most reasonable way to achieve the estimation

accuracy, reliability and versatility demanded by real world missions.

In this section, we describe a novel heterogeneous sensor fusion method. We build upon the EKF framework and LO described in Chapter 4 by adding two pose estimation methods: a VO module, presented in [64], and an ICP-based LiDAR localization method, presented in [101].

The work presented in this section is the result of a collaboration with the Robot Perception Group, led by Dr. Maurice Fallon [1].

5.5.1 Visual Odometry

The VO method presented in [64] was originally developed for drones equipped with an IMU and the Kinect depth sensor. In this section, we evaluate its performance with the depth data provided by the stereo camera of the Multisense SL sensor. In contrast to depth sensors, the quality of the point clouds generated with stereo cameras is highly dependent on how much texture is present in the scene. Hence, dark or textureless areas can lead to poor results. On the other hand, depth sensors can work in total darkness, but are prone to failure when sunlight interferes with the light beams projected by the device. The algorithm is used to estimate frame-to-frame rigid transformations, at 10 Hz. It consists of several steps, including:

Image preprocessing: conversion to grayscale, Gaussian smoothing, Gaussian pyramid to identify strong features at multiple scale levels;

Feature extraction: a FAST corner detector [116] is applied to each pyramid level, and only the best scoring feature descriptors are retained;

Rotation estimation: an initial guess for the rotation is estimated through analysis of current and previous frames;

Feature matching: each feature is normalized and assigned to a descriptor. The matching between the features in the stereo pair is done through minimization of the SAD metric and further optimization for refinement;

Inlier detection: the matches between features and their relative distances are arranged as vertices and edges of a graph, respectively. A greedy algorithm is used to approximate the maximal clique of the graph, to reject weak correspondences;

Motion estimation: the rigid transformation estimate is obtained by minimization of the Euclidean distances between inliers, and then by minimizing the reprojection error with a non-linear least-squares solver. Finally, a keyframe technique is used to reduce short scale drift.

For further information we invite the reader to consult the original work in [64].

The rigid transformations produced by the algorithm are integrated into the EKF as position (linear and angular) corrections.

[1] http://robotperception.inf.ed.ac.uk/

5.5.2 Auto-tuned ICP

The Auto-tuned ICP method was recently introduced in [101] to overcome the limitations of the GPF and baseline ICP-based methods, which are prone to failure in the presence of variable overlapping between frames to be registered, and to clutter. The method computes online a metric Ω that indicates the amount of overlapping between the *source* and *target* point clouds, and automatically adapts the algorithm in accordance with that measurement. Before performing the registration, the cloud is preprocessed to include only planar features – an opposite approach to what was presented in Section 5.2. This is well justified by the large FoV provided by the LiDAR mounted on the Multisense SL sensor of Carnegie Robotics. Nominally, it can cover 75% of a sphere after a complete rotation. By comparison, a depth sensor with a FoV of $58^\circ \times 45^\circ$ covers only 4%.

The algorithm builds on the other modules by sending periodical position corrections at low frequency ($0.5\,\mathrm{Hz}$). Between two corrections, the algorithm accumulates scans using the poses computed by the EKF from the other sensor modalities, and computes the registration.

5.5.3 Fusion Algorithm

Both the VO module and the AICP algorithms provide asynchronous pose updates to the EKF filter. This requires a way to avoid conflicts between the updates and to handle delays, which can be in the order of seconds. The filter keeps a history of measurements. Whenever a new update is available, the history is rewritten from the timestamp associated with the update until the most recent event. This allows it to handle unordered, concurrent updates from the ICP and VO processes. An example is provided in Figure 5.13: at event #1 a new LiDAR correction becomes available. Its timestamp is in between the history of poses computed by the inertial and kinematics processes. At event #2 the correction is incorporated into the filter state, and the portion of history in green is rewritten. At event #3 the filter has propagated to the corrected state, and a new VO update in the past becomes available. At event #4 the update is incorporated, and the portion of history in green is again rewritten.

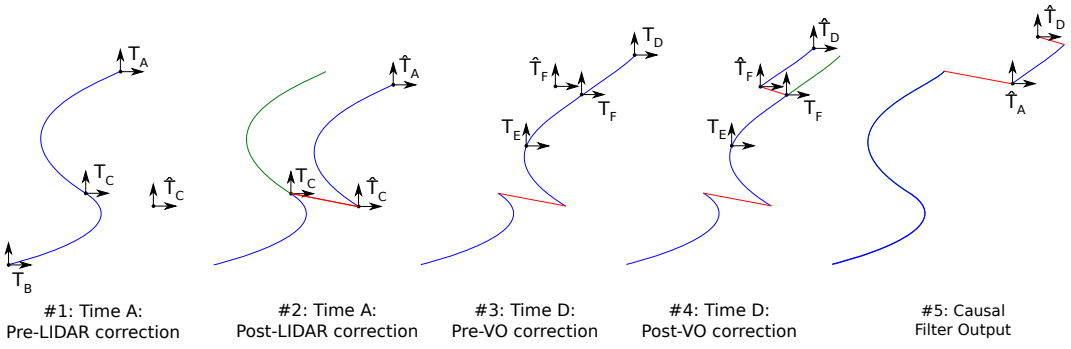


Figure 5.13: Example of concurrent pose correction from LiDAR and Visual Odometry.

5.5.4 Experimental Results

In this section, we show the performance of the complete fusion algorithm running on HyQ in challenging conditions (see Appendix A.3). We validate the pose estimation performance both outside and inside the control loop.

Pose Estimation Performance

With a real mission in mind, we evaluated the filter performance on a challenging dataset recorded (with the robot in Configuration A) in a $17\,\mathrm{m}$ long semi-structured environment (see Appendix A.3). The recorded motions executed on HyQ included trotting at different speeds and directions, sharp turns, and crawling on uneven and rough terrain. The level of difficulty was raised further by the poor lighting conditions and occlusion caused by the protection bar of the Multisense SL sensor (see Figure A.2).

Despite all these conditions, the algorithm was able to successfully reconstruct the scene. The different sensor modalities are seamlessly fused to achieve a smooth position signal. In particular, Figure 5.14 shows that:

- the combination of IMU and kinematics provides a smooth but drifting estimate (cyan line);
- the combination of IMU and VO also provides a smooth but drifting estimate (magenta line);
- the combination of IMU, LO and AICP provides a non-drifting yet discontinuous estimate, due to low frequency updates (yellow line);
- the combination of all of the above signals provides a smooth and non-drifting estimate, due to the contribution of a hierarchy of frequency updates (green line): inertial and kinematics at $1\,\mathrm{kHz}$, VO at $10\,\mathrm{Hz}$ and AICP at $0.5\,\mathrm{Hz}$.

Note that, in order to work, the AICP requires the availability of a high frequency pose estimate between the completion of two pose corrections. The pose is needed to properly accumulate the various scans acquired at different angular positions of the sensor head. If the filter pose is drifting too much during this time interval, the accumulated point clouds will be distorted, and the registration will fail.

The hierarchy is also fundamental for fault tolerance: the core of the estimator (*i.e.,* the proprioceptive-based estimation described in Chapter 4) runs at high frequency and is totally decoupled from the registration-base pose corrections provided by the AICP module. If the correction is unavailable for some reason, *e.g.,* because of a sensor failure, the estimate will degrade gracefully because the filter continues to use the proprioceptive inputs to compute the state. In other words, the AICP, which runs at low frequency, depends on LO – but the opposite does not hold.

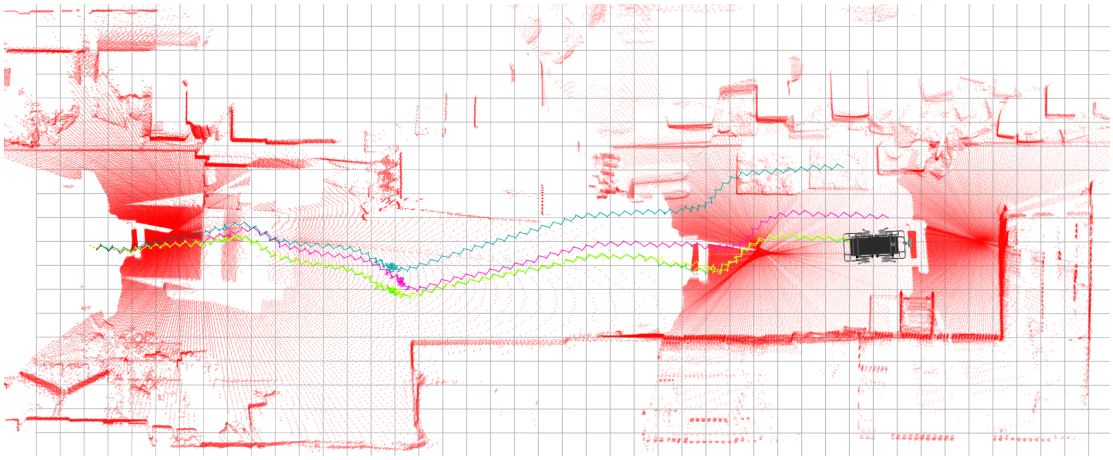
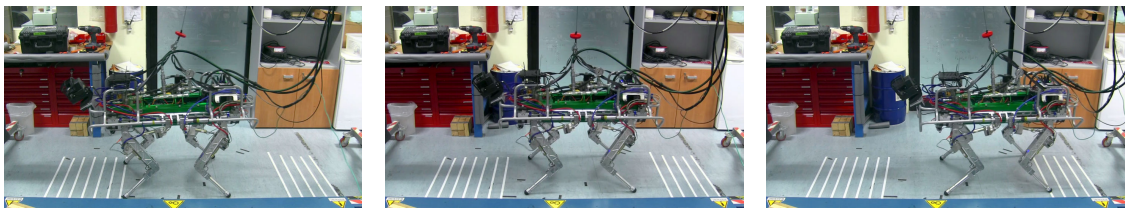


Figure 5.14: Estimated trajectories with different sensor modalities: inertial + kinematics (cyan); inertial + visual odometry (magenta); inertial + kinematics + registration (yellow); inertial + kinematics + visual odomeetry + registration (green). In red, the test area as reconstructed using all the modalities.



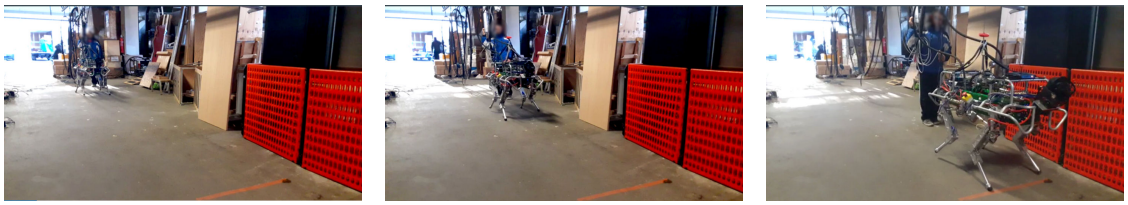
(a) **(b)** **(c)**

Figure 5.15: Pose Estimation in-the-loop: indoor square wave experiment. The robot is commanded to move from one white stripe on the left to one white stripe on the right multiple times, at increasing speeds and distances. The stripes are placed $10\,\mathrm{cm}$ apart. After $15\,\mathrm{min}$ of motions, the robot is still correctly stopping at the chosen white line.

Control In-the-Loop Performance

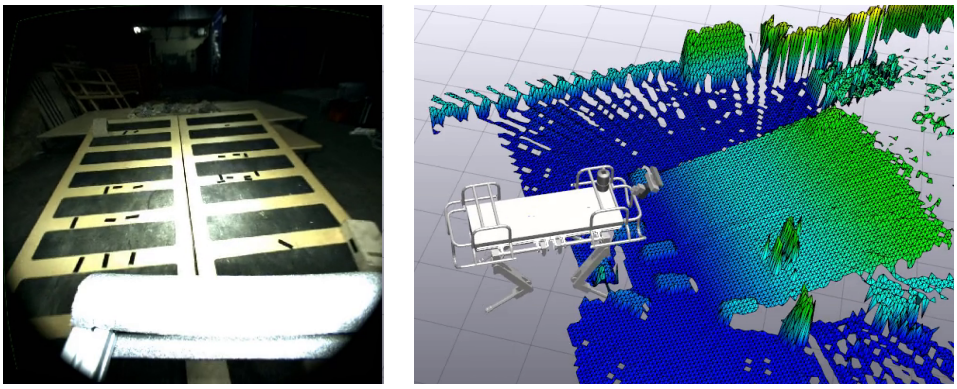
The performance of the filter update is further evaluated by direct use in the control loop. We designed a square wave experiment where the robot is commanded to oscillate between two lines several times at increasing speeds (see Figure 5.15). At every speed increment, a wider interval is chosen. After more than 50 periods and a speed increment from $0.2\,\mathrm{m/s}$ to $0.6\,\mathrm{m/s}$, the feet still land at the desired location, indicated by the white stripes on the floor in Figure 5.15.

As a further demonstration of its robustness, the algorithm was executed in a similar experiment but in a different environment. The robot's motion was repeated over a longer distance ($5\,\mathrm{m}$) with no issues (see Figure 5.16).



(a) (b) (c)

Figure 5.16: Pose Estimation in-the-loop: outdoor square wave experiment.The robot is commanded to move from the starting position to the orange stripe, at $5\,\mathrm{m}$. After more than 10 of forward-backward motions (*i.e.,* $50\,\mathrm{m}$ traveled), the robot is still correctly stopping at the orange line.



(a) (b)

Figure 5.17: Example of online LiDAR mapping with multi-sensor state estimation. **a)** visual feedback from the Multisense SL stereo camera. Note the difficult conditions of the scene: underexposed areas due to night operations, overexposed areas due to light flashing, occlusions due to the protection frame, and image blur due to shocks. **b)** map reconstructed in realtime from LiDAR scans. Note that, since the LiDAR source is a rotating planar device, the map is build scan-by-scan using the state estimator. Note also how the bricks are accurately reconstructed

Online Mapping

One of the fundamental applications of accurate state estimation (and in particular pose estimation) is mapping. Figure 5.17 provides an example of online mapping with LiDAR scans. Even though the LiDAR has better performance than cameras, it is much sparser, because it provides only a planar set of points. Therefore, the state estimator has to provide a smooth high frequency pose estimate, in order to collect each scan and fuse it with the previous. Despite the adverse lighting conditions (Figure 5.17a), the bricks and the ramp in front of the robot are correctly reconstructed (Figure 5.17b). This result highlights the benefit of having multiple exteroceptive sources for the pose estimation, and demonstrates the quality of the presented algorithm.

Method	Limitations
FRSM [5]	2D planar motion required
Selective ICP [26]	reduced FoV
GPF [22]	accurate prior map required
AICP [101]	multiple planar regions required
FOVIS [64]	textured scenes required

Table 5.1: Limitations of the exteroceptive pose estimation methods.

5.6 Conclusion

Accurate pose estimation is a fundamental component of a legged navigation system, since it enables the safe execution of both foot and body trajectories. Furthermore, it allows for the creation of accurate geometrical representations of the environment, as we will see in the next chapter. Good representations of the environment constitute an essential input for foothold and body trajectory planners, which are used to overcome difficult areas.

In this chapter, we have shown several pose estimation algorithms which individually display good localization performance in controlled environments. In Section 5.2 we have presented a new efficient selection algorithm for ICP, which reduces the computational burden and increases the accuracy when the scene lacks of feature for the registration. The work has been published in [26] as first author. In Sections 5.3 and 5.4 we have studied the performance of two state-of-the-art LiDAR localization algorithms on HyQ: the FRSM [5] and the GPF [22]. The two methods are fused with IMU and LO measurements using the filter presented in Chapter 4. Originally developed for flying robots, only one the two methods have been tested on a legged machine [37], yet it has never been executed inside the control loop during dynamic motions.

Despite the good performance shown by these methods, in a field mission no one method alone can generalize all possible scenarios.

Table 5.1 lists the different approaches shown, together with their main "weakness". Throughout the chapter, we have summarized the progress towards a versatile, robust and general way to estimate a pose on HyQ robots. We also have proven that a hierarchical sensor fusion approach (*i.e.,* from high frequency drifting updates to drift-free low frequency updates) with multiple pose estimators can generalize well against the adverse lighting and motion conditions a real scenario is expected to have. Section 5.5 showed the most recent results in this direction. We envision this approach to be the most versatile, because it can handle challenging situations by leveraging both 2D and 3D features, as well as kinematics and inertial navigation. To date, the most similar results to what presented in Section 5.5 have been presented in [83], where good pose estimation performance is shown, yet in-the-loop demonstration is missing.

Future ways to improve this work include the incorporation of more sophisticated methods of performing concurrent integration of multiple pose updates, and the integration of more sophisticated algorithms for VO.

Chapter 6

Robocentric Mapping and Locomotion Applications

In the previous chapter, we presented several algorithms for accurate pose estimation of dynamic legged robots. Pose estimation is fundamental for effective tracking of base and feet trajectories, but it is also essential to build, online and onboard, a consistent geometrical representation of the environment. We refer to this process, simply, with the term *mapping*. In this chapter, we discuss what mapping approaches are suitable for the locomotion of dynamic legged robots. In particular, we are interested in mapping algorithms for reactive or short-term planned locomotion, rather than global path planning navigation.

Short term planning (*i.e.,* planning of base and feet trajectories with a horizon of 4–5 steps) is motivated by biology [150]: studies on human subjects demonstrated that, for safe locomotion on difficult terrains, two steps planned ahead are sufficient to achieve stable locomotion in almost every condition, despite disturbances and difficulty of the terrain. To generate appropriate body trajectories, the robot needs a small scale (2–3 robot lengths) but detailed (≈ 1–$2\,\mathrm{cm}$ accuracy) map of its surroundings.

In contrast, global path planning for long-term navigation usually has different requirements. Over long distances (tens to thousand of meters), we need techniques to build a compact representation of large scale areas. Moreover, for such distances, even a small drift in the pose estimate eventually becomes large and needs to be corrected. This problem is generally solved with topological maps, graph based localization, and loop-closure detection. These and other techniques are part of a large field of study: the SLAM theory [51].

The focus of this chapter is small scale mapping and its use of endowing HyQ with sophisticated locomotion skills, such as walking on stepping stones or promptly avoiding obstacles on-the-fly while trotting.

The remainder of this chapter is divided as follows: Section 6.1 is dedicated to the definition of mapping perspectives and the implementation of different representations on HyQ; in Section 6.2 we describe a novel method based on terrain pattern classification to reactively negotiate obstacles while trotting; in Section 6.3 we describe an image processing approach to costmap

computation for foothold planning; in Section 6.4 we discuss strengths and limitations of the proposed approaches; the chapter is concluded by Section 6.5 where we summarize and indicate future directions.

I published the work presented in this chapter in [26] as first author, and [9, 11] as second author.

6.1 Mapping Representations

In this section, we analyze the difference between *global mapping* and *robocentric mapping*, and we describe two mapping algorithms (one 3D and one 2D) implemented on HyQ.

6.1.1 Global Mapping vs. Robocentric Mapping

Pose estimation algorithms based on point cloud registration (*e.g.,* ICP) compute, at each time step, the rigid transformation which aligns the most recent point cloud to a past cloud, which is used as a reference. The transformations are then incrementally accumulated in order to estimate the rigid transformation which aligns the latest point cloud available with the first point cloud acquired. By applying the appropriate static transformations (*i.e.,* between camera and robot base), this is the equivalent of estimating the rigid transformation between the world frame and the base frame, *i.e.,* the pose of the robot. If during this process we accumulate the aligned point clouds, implicitly, we have also implemented a mapping algorithm.

Let us define a point cloud $\mathcal{C}_k \in \mathbb{R}^{N\times 3}$, captured at time k by a camera or a LiDAR sensor. The points of this cloud all refer to the optical frame of the device. Since the sensor is moving together with the robot, but the captured scene is static, the points of a new point cloud are defined in a new frame of reference. When not ambiguous, we will identify all these new frames of reference with their corresponding time step, k. The pairwise registration algorithm provides an estimate of the rigid transformation ${}_{k-1}\mathrm{T}_k$, which aligns the current cloud $\mathcal{C}_k$ to the previous cloud $\mathcal{C}_{k-1}$. With these elements, we can recursively compute the current map $\mathcal{M}_k$, defined in the frame of the first cloud $\mathcal{C}_0$, as:

$$ {}_0\mathrm{T}_k = {}_0\mathrm{T}_{k-1}\,{}_{k-1}\mathrm{T}_k = \left(\prod_{i=0}^{k-1} {}_i\mathrm{T}_{i+1}\right) \tag{6.1} $$

$$ \mathcal{M}_k = \mathcal{M}_{k-1} + {}_0\mathrm{T}_k\mathcal{C}_k \tag{6.2} $$

with the initial condition $\mathcal{M}_0 = \mathcal{C}_0$.

Equation 6.1 incrementally builds a map of the environment $\mathcal{M}_k$. All the points of the map are the sum of the clouds collected over time and aligned with the first cloud $\mathcal{C}_0$. Since all the points from the map are expressed in the same frame of reference (that of $\mathcal{C}_0$) we can arbitrary choose it as our *world frame*. We call this mapping procedure *global mapping*, since all the points are expressed in an inertial, fixed frame.

The close form of Equation 6.1 is given by:

$$\mathcal{M}_k = \mathcal{C}_0 + \sum_{i=0}^{k-1} \left[\underbrace{\left(\prod_{j=0}^{i} {}_j\mathrm{T}_{j+1} \right)}_{={}_0\mathrm{T}_{i+1}} \mathcal{C}_{i+1} \right] \tag{6.3}$$

Equation 6.3 shows that as i increases (*i.e.,* the closer we get to the most recent cloud, $\mathcal{C}_k$) more terms are accumulated in the transform ${}_0\mathrm{T}_{i+1} = {}_0\mathrm{T}_1 \dots {}_i\mathrm{T}_{i+1}$. Since each term ${}_i\mathrm{T}_{i+1}$ is estimated by the registration algorithm, it is affected by a certain amount of error. Therefore, the largest error is accumulated around the most recent cloud $\mathcal{C}_k$.

Global mapping is not really needed for foothold planning. Events far away in space should not affect the decisions for steps taken in the near future. Hence, the localization accuracy should be inversely proportional to the age of the data. We define the following algorithm:

$$\mathcal{M}_k = \mathcal{C}_k + {}_k\mathrm{T}_{k-1}\mathcal{M}_{k-1} \tag{6.4}$$

where the initial condition is again $\mathcal{M}_0 = \mathcal{C}_0$. In this case, at every step k, the current map $\mathcal{M}_k$ is given by the sum between the most recent cloud $\mathcal{C}_k$ and the previous map $\mathcal{M}_{k-1}$, aligned with $\mathcal{C}_k$.

We define this approach as *robocentric mapping*, because at each time step k the current map $\mathcal{M}_k$ is aligned with the current sensor frame, which is rigidly attached to the robot.

The close form of Equation 6.4 is:

$$\mathcal{M}_k = \mathcal{C}_k + \sum_{i=0}^{k-1} \left[\underbrace{\left(\prod_{j=k}^{k-i} {}_j\mathrm{T}_{j-1} \right)}_{{}_k\mathrm{T}_{k-i-1}} \mathcal{C}_{k-i-1} \right] \tag{6.5}$$

Equation 6.5 shows that the expansion of the product ${}_k\mathrm{T}_{k-i-1} = {}_k\mathrm{T}_{k-1} \dots {}_{k-i}\mathrm{T}_{k-i-1}$ has less terms when the newest clouds are accumulated.

Geometrically, the maps generated by Equation 6.3 and 6.5 only differ by a rigid transformation. In the first case, the map is the sum of clouds aligned to the first one, $\mathcal{C}_0$, while in the second, all the clouds are expressed in the frame of the most recent one, $\mathcal{C}_k$. Hence, the drift is the same for both maps, but it is distributed differently. Assuming a continuous motion, with the robocentric mapping approach the error due to drift is concentrated far from the robot, and the most accurate portion of the map (*i.e.,* the one used by the planner/controller) is close to it.

The advantage of robocentric mapping is shown by Figure 6.1: the robot is approaching a horizontal obstacle on flat ground. The point cloud colormap is proportional to the z-axis coordinate: low areas start from red and high areas go to violet. Due to a small pitch error accumulated over time, the terrain behind the robot appears lower than in reality. With global mapping, the opposite situation would happen, causing an underestimation of the obstacle's height.

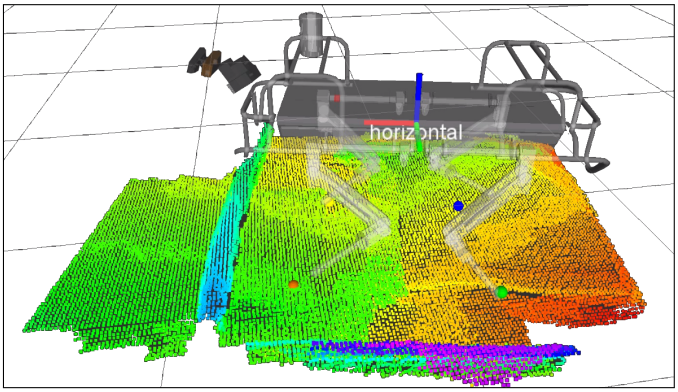


Figure 6.1: Example of local mapping on a flat terrain with an horizontal obstacle. The colormap follows the z-axis and ranges from red (low values) to blue/purple (high values). True ground level is green. Note that the rear part of the map (on the right) is older and drifted downwards on the z-axis

6.1.2 Robocentric Point Cloud Mapping

In the previous section, we have shown the advantage of using robocentric mapping over global mapping. In this section, we describe the implementation of a robocentric point cloud mapping algorithm, suitable for reactive and short-term planned locomotion of the HyQ robot.

Algorithm 3 Robocentric Point Cloud Mapping

1: $\mathcal{M}_0 = {}_b\mathrm{T}_c\mathcal{C}_{c,0}$
2: **for each** $k > 0$ **do**
3: $\quad\mathcal{C}_k \leftarrow {}_b\mathrm{T}_c\mathcal{C}_{c,k}$ ▷ Cloud into base frame
4: $\quad\mathcal{M}_k \leftarrow {}_k\mathrm{T}_{k-1}\mathcal{M}_{k-1}$ ▷ Map into current base frame
5: $\quad$**for each** $\mathbf{p} \in \mathcal{M}_k$ **do**
6: $\quad\quad$**if** ($p_x < \min_x \mathcal{C}_k$ **or** $p_y < \min_y \mathcal{C}_k$**or** $p_y > \max_y \mathcal{C}_k$) **and**
7: $\quad\quad\ |p_x| < \theta_x$ **and** $|p_y| < \theta_y$ **then** ▷ $\mathbf{p}$ outside $\mathcal{C}_k$ and within limits
8: $\quad\quad\quad\mathcal{C}_b \leftarrow \mathcal{C}_b + \mathbf{p}$
9: $\quad\quad$**end if**
10: $\quad$**end for**
11: $\quad\mathcal{M}_k \leftarrow \mathrm{voxelf}(\mathcal{C}_k, v_{\text{size}})$
12: **end for**

The pseudocode is listed in Algorithm 3. First, we initialize the map with the first point cloud transformed into the base frame (line 1). Then, given a new cloud $\mathcal{C}_{c,k}$ expressed at time k and in the camera (or LiDAR) frame c, we transform it into the actual base frame b. In the following, we omit the subscript for the base frame, and call the result simply $\mathcal{C}_k$ (line 3). Without loss of generality, we assume that the pose estimation algorithm provides the incremental transformation ${}_k\mathrm{T}_{k-1}$ between the base frame at time $k-1$ and the base frame at time k. In the case of a pure ICP-based estimator, such as the one described in Section 5.2, the output is already incremental. Hence, we can apply the static transformation between camera (LiDAR) frame and base frame in order to compute it. If the estimator is giving an absolute pose,

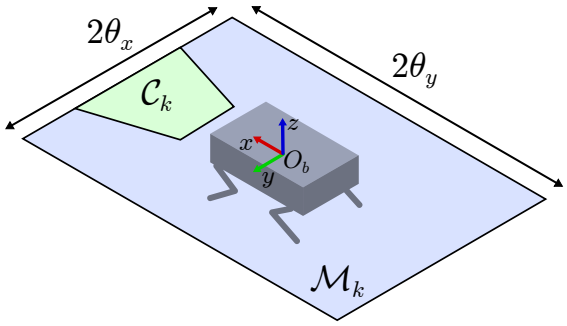


Figure 6.2: Robocentric map example. The current map $\mathcal{M}_k$ is the sum of the blue and green areas. Points exceeding the blue area are trimmed to keep the map compact.

as in the EKF of Section 4.1, we can compute the incremental transformation from the two poses recorded at time k and $k-1$, as: ${}_k\mathrm{T}_{k-1} = ({}_w\mathrm{T}_k)^{-1}({}_w\mathrm{T}_{k-1})$.

We apply ${}_k\mathrm{T}_{k-1}$ to the previous map $\mathcal{M}_{k-1}$, as in Equation 6.4 (line 4). Then, we add to $\mathcal{C}_{b,k}$ only the points $\mathbf{p} \in \mathcal{M}_k$ that are outside the border of $\mathcal{C}_k$, which is indicated by the green area in Figure 6.2 (line 6). With this process, we want to discard all the points in the map which are overlapping with the FoV of the camera, with some tolerance to account for the trapezoidal shape of the projection of $\mathcal{C}_k$ on the ground. At the same time, we also discard all the points of the map which are outside a predefined $2\theta_x \times 2\theta_y$ area that surrounds the robot (light blue area of Figure 6.2). This can easily be done because all the points are expressed in the base frame of reference (line 6). Finally, the map is passed through a voxel filter, in order to equally distribute the points (line 11).

The boundaries of the map $(2\theta_x, 2\theta_y)$ and the voxel size v_{size} are crucial parameters for the system: small voxels (*e.g.,* $< 1\,\mathrm{cm}$) would make the cloud too dense to be processed in time (*i.e.,* less than the sampling period), while a sparse cloud would produce gaps in the heightmap extracted from it (see Section 6.1.3). Similarly, large boundaries would reduce the output frequency of the mapper. For our experimental setup, we used a voxel size of $2\,\mathrm{cm}$, and a map $2\,\mathrm{m}$ wide and $3.5\,\mathrm{m}$ long, with the robot center placed $0.25\,\mathrm{m}$ in front of the map center. These parameters met the desired update frequency of $30\,\mathrm{Hz}$, with the depth sensor from Configuration A (see Chapter 3 for more details on the sensing equipment).

The robocentric point cloud mapping algorithm described above allows us to keep a small but detailed tridimensional model of the terrain. The memory allocated for the map is fixed over time, because any data outside the region of interest is discarded as the robot moves. This is an important advantage for online foothold planning, because the computation of features for the optimization is performed only where needed. However, when a simpler representation is sufficient, *e.g.,* when we are not interested in overhanging objects or tunnels, elevation maps are preferred, because they are dimensionally less expensive.

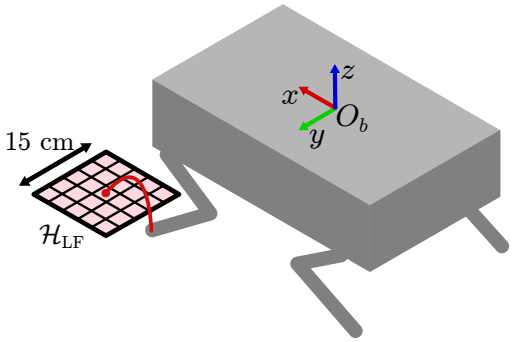


Figure 6.3: Local heightmap example. The heightmap is computed on the desired foothold of the LF leg, with $d = 5$ and $r = 3\,\text{cm}$

6.1.3 Elevation Mapping

An *elevation map*, or *heightmap*, is a matrix $\mathcal{H}_k \in \mathbb{R}^{n \times m}$ whose values indicate the average distance from an arbitrary plane. Typically the xy-plane of the base is used, but a gravity-aligned plane can also be useful for trajectory generation, as we see in Section 6.2.

A heightmap can be extracted from a point cloud map with the following steps:

1. project the point cloud map onto the plane of choice;
2. divide the plane into a regular grid of cells;
3. compute the average height of the points within the same cell.

Note that, with this procedure, we lose the third dimension and possibly generate undesired artifacts. Let us consider two different objects stacked on top of each other (*e.g.*, a table top and the floor). Their heightmap counterpart would be merged into one solid object with half the size of the tallest of the two stacked objects. This can lead to failures if the robot plans to climb the artifact. To avoid this issue, we can trim the original point cloud to equal the maximum stretch of the legs, and/or use a different aggregation function to merge the points within the same cell, such as the maximum or the median. Another problem when converting from point clouds to heightmaps is gaps. To avoid gaps, the size of the grid cells should be bigger or equal to the voxel size used to filter the point cloud map.

Local Heightmaps

In some circumstances, there it is not necessary to compute the entire heightmap map from the point cloud map, but only in the vicinity of the footholds, as depicted in Figure 6.3. Given a desired foothold position, the corresponding heightmap is computed in the same way as the full point cloud. However, this time we also discard all the points outside a square area of $d^2 \cdot r^2$ around the desired foothold location, with d and r being the number of cells per line and the width of each cell in meters, respectively. When no points are available, the predicted z location of the foothold can be used.

The use of local heightmaps around footholds is useful to speed up the computation. The robot keeps a point cloud of its vicinity, and computes the heightmap on demand, where requested. Again, the proper choice of the parameters d and r is crucial to keep real-time performance.

In the next section, we see a real scenario application of local heightmaps to enable obstacle avoidance on-the-fly while trotting.

6.2 Terrain Pattern Classification for Visual Reactive Trotting

The RCF[8] is a robust central pattern generation controller. It has been proven to be effective against disturbances and small obstacles when walking blindly. However, to overcome or avoid obstacles which are higher than a few cm, we need visual feedback. Let us consider a collection of four local heightmaps (one per foothold), defined to be parallel to the xy-region of the horizontal frame (see Section 3.2). We wish to select, from these heightmaps, the landing position which minimizes foot and shin collisions with frontal obstacles.

For this task we set the heightmaps parameters to $d = 15$ and $r = 2\,\mathrm{cm}$. However, our heightmap extraction software is designed to change these values in real-time. Since the feet trajectories are generated in the horizontal frame, the plane where heightmaps are generated is the xy-plane of the horizontal frame. At the beginning of each step, the RCF controller sends the desired foothold location to the mapper and waits for the corresponding local heightmap until the apex of the foot swing trajectory is reached. To reduce the effect of noise, a Gaussian Filter of size 3×3 pixels is also applied to the heightmap before it is sent.

6.2.1 Foothold Correction as a Classification Problem

Once the heightmap is received, we need to compute the optimal foot touchdown position. We can consider the local heightmaps as matrices, and solve this problem from a machine learning perspective.

In particular, we treat the obstacle negotiation as a supervised classification problem: each heightmap is a feature vector $\mathbf{x} = [1\ x_1\ \cdots\ x_n]^{\mathsf{T}} \in \mathbb{R}^{(n+1)\times 1}$, where n is the number of cells of the heightmap, and an extra term is added to include the bias term. The class $y \in \mathbb{N}^m$ indicates one of the m possible corrections applicable to that nominal foothold position (see Figure 6.4).

To maintain a trade-off between speed and accuracy, we opted for heightmaps composed of 15×15 cells, and for an output set of 8 possible displacements, one for each cardinal direction, plus the "null" displacement. Hence, $\mathbf{x} \in \mathbb{R}^{226\times 1}$ and $y \in \mathbb{N}^9$.

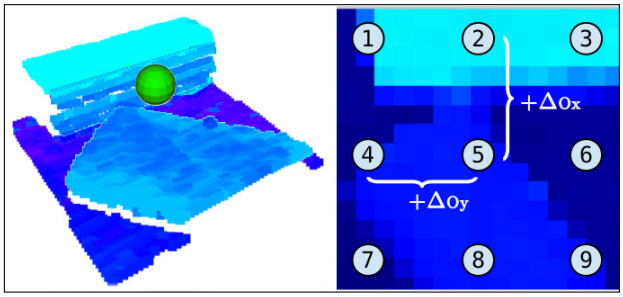


Figure 6.4: *Left*: example of point cloud in colorized by elevation (from dark blue to cyan). The cloud covers an area of $30 \times 30\,\mathrm{cm}$. The green ball indicates a desired foothold position. *Right*: the corresponding heightmap with the 8 possible footholds adjustments and the displacements according to the robot axis convention.

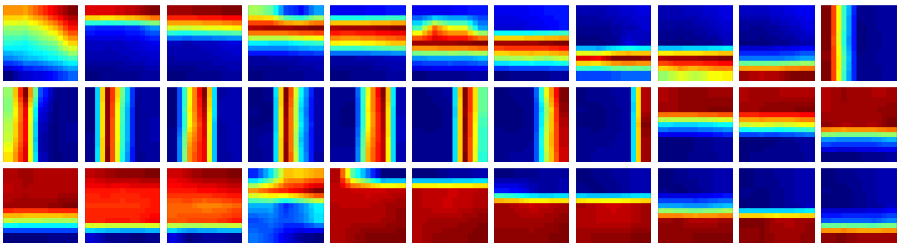


Figure 6.5: Examples of pattern, taken from the training set. The images are in false colors and rescaled between minimum (dark blue) and maximum (dark red) values of the heightmap pattern. Each pixel represents the average height of a 2 cm^2 area of terrain, referred to the horizontal frame of the robot.

6.2.2 Training Set Generation

We generated a dedicated training set for each leg. First, we defined 33 different patterns that fit a variety of possible obstacles, namely: stairs, bars, logs or stones. An example of the 33 categories is depicted in Figure 6.5.

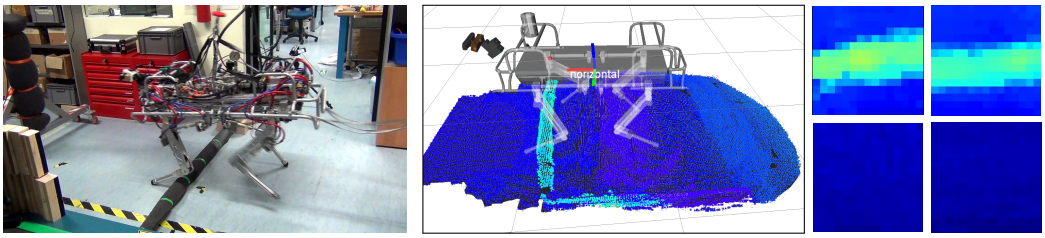
For each pattern, we acquired 100 samples from real depth sensor imagery, for a total of 3300 input examples. Since the type of trajectory adjustment depends on the involved leg, the set of examples was replicated four times and independently labeled for each leg.

6.2.3 Logistic Regression for Foothold Decision

As a classifier, we opted for a logistic regressor [55], which had been demonstrated to be sufficiently robust (90% success rate) and fast (only four 15×15 matrix multiplications) during our preliminary tests with the patterns shown in Figure 6.5.

Since we have more than one class (in our case $m = 9$), we opted for a One-vs-All multi-class classification by training m binary classifiers h_{θ_i}, $i \in \mathbb{N}^m$ where $\theta_i \in \mathbb{R}^{(n+1)\times 1}$ is the weight vector for the i-th class and:

$$h_{\theta_i} = h_{\theta_i}(\mathbf{x}) = \frac{1}{1 + \exp\left(-\theta_i^{\mathrm{T}}\mathbf{x}\right)} \tag{6.6}$$



(a) robot **(b)** point cloud map **(c)** local heightmaps

Figure 6.6: Log traversal experiment. The robot is crossing a horizontal obstacle made of rolled soft material simulating a fallen log. **(a)** the robot is adjusting the nominal foot position to step in front of the obstacle. **(b)** the point cloud map at the time of crossing. **(c)** the four heightmaps at the nominal foot positions. The obstacle is clearly visible by the two front legs (top left and right squares).

Given a specific leg, a class $c \in \mathbb{N}^m$ and a training set of k examples $(\mathbf{x}^{(i)}, \tilde{y}^{(i)})$, with $\mathbf{x}^{(i)} \in \mathbb{R}^{(n+1)\times 1},\ i \in \mathbb{N}^k$, and $\tilde{y} \in \{0, 1\}$ (which indicates whether $y = c$ or not) we compute the weight vector θ_c by minimizing the cost function:

$$J(\theta_c) = +\frac{1}{k}\sum_{i=1}^{k} C(h_{\theta_c}(\mathbf{x}^{(i)}), \tilde{y}^{(i)}) + \lambda \sum_{j=1}^{n} (\theta_c^j)^2 \tag{6.7}$$

where:

$$C(h_{\theta_c}(\mathbf{x}^{(i)}), \tilde{y}^{(i)}) = \log\left(h_{\theta_c}(\mathbf{x}^{(i)}\right) + (1 - \tilde{y}^{(i)}) \log\left(1 - h_{\theta_c}(\mathbf{x}^{(i)})\right) \tag{6.8}$$

and $\lambda = 0.001$.

From the weight vectors we define the weight matrix of a leg as $\Theta_{\text{leg}} = [\theta_1\ \theta_2\ \theta_c\ \ldots\ \theta_m]^{\mathrm{T}} \in \mathbb{R}^{(n+1)\times m}$ and $\mathbf{h}_{\Theta_{\text{leg}}}(\mathbf{x}) = [h_{\theta_1}\ h_{\theta_2}\ \cdots\ h_{\theta_m}]^{\mathrm{T}}$.

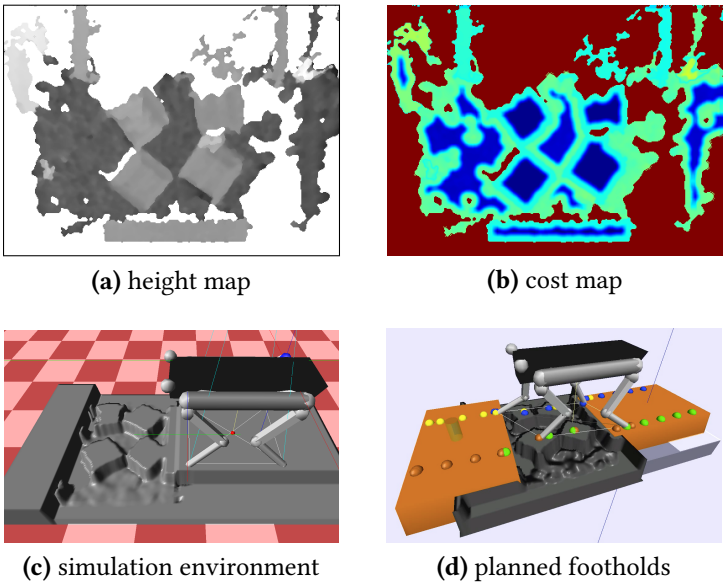
The predicted class y for a test example $\mathbf{x}_t$ is then computed as the index of:

$$\max\left(\mathbf{h}_{\Theta_{\text{leg}}}(\mathbf{x}_t)\right) \tag{6.9}$$

We assigned to each class y a corresponding offset $(\Delta_{Ox}^{\text{leg}}, \Delta_{Oy}^{\text{leg}})$, as depicted in Figure 6.4.

6.2.4 Experimental Results

We validated the performance of this approach with a series of experiments on HyQ. We placed a horizontal obstacle (*e.g.,* a rolled carped similar to a log) with a height of 8–$12\,\mathrm{cm}$ in front of the robot and crossed it with the robot trotting forwards and backwards. Figure 6.6 captures the instant when the robot is crossing the obstacle. In Figure 6.6a we see the robot successfully adjusting the step length and place its foot after the obstacle. Figure 6.6b shows the point cloud map at the same moment, while 6.6c shows the four heightmaps extracted at the desired footholds. The obstacle is clearly visible by the two front legs (top left and right squares), and was correctly classified (*cf.* with sixth and seventh patterns in the top row of Figure 6.5).



(a) height map **(b)** cost map

(c) simulation environment **(d)** planned footholds

Figure 6.7: Foothold planning results using a heightmap

In the next section, we describe another application of elevation maps for quadrupedal locomotion.

6.3 Terrain Cost Map for Foothold Planning

A common approach for foothold planning from elevation maps $\mathcal{H}_k \in \mathbb{R}^{n \times m}$ is to associate a traversal cost to each value of the map. Given this costmap, the most suitable sequence of footholds to reach a goal can be computed as the optimal sequence which minimizes the overall cost of traversal [137, 41]. In this section, we describe how to compute a costmap from a heightmap with image processing techniques.

From an image processing perspective, a cost map for foothold planning should assign a high cost to the following features in the heightmap:

High frequencies: they correspond to discontinuities on the morphology, such as the edges of a rock;

Small uniform areas: since the robot has a non negligible foot area and it could miss such foothold;

Extreme values: in particular, the relative height from the terrain to the nominal position of the leg ($56\,\mathrm{cm}$ from the base). The maximal extension and retraction for a leg are $10\,\mathrm{cm}$ and $20\,\mathrm{cm}$, respectively. Beyond these values, the cost should be infinite, since it is out of the leg workspace.

To incorporate this information into the costmap, we compute the image derivatives of the elevation map, and then we assign a cost which depends both on the direction and the intensity

of those derivatives. The cost map computation involves five steps:

Pre-processing: the elevation map is first processed by a sharp filter, to ease the process of edge detection;

Edge detection: the edges of a height map reflect the actual discontinuities of the ground. To compute them we apply a Canny edge detector [28];

Morphology operator: since the foot has a finite area, we want to avoid the margins around the edges. We apply a dilation operator with a round structuring element along the edges. The size of the structuring element is chosen according to the dimension of the foot and the desired safety margin for the planning;

Gaussian filtering: we apply a Gaussian filter to incorporate uncertainties around the edges. The covariance is chosen in accordance with the uncertainty of the mapping, pose estimation, and foot trajectory tracking;

Height cost computation: the cost map computed in the previous steps is linearly combined with the original height map, to take into account the cost of the height of the cells.

Figure 6.7a shows an example of a heightmap. The heightmap has been computed by reprojection of the point cloud extracted from a Bumblebee stereo camera as described in [10]. The corresponding costmap is shown in Figure 6.7b. In this figure, the values of the colormap vary between dark blue for low costs and dark red for high costs. For the final step of the costmap computation (height cost), we compute the absolute difference between the robot's foot level and the height map. This can be noted by looking at the center of stepping stones, which have the smallest cost because they lie at the same level of the robot's feet (see Figure 6.7c). Unknown heights were treated as a special case by assigning them to the maximum cost (dark red). The final planned foothold sequence from the costmap is shown in Figure 6.7d [11].

6.4 Discussion

In Section, 6.1 we have demonstrated how a robocentric approach to mapping is more suitable for locomotion applications, where loop-closure techniques are not required. Its formulation is equivalent to the more commonly used global mapping, but it better distributes the drift over the map. The only potential drawback of this solution is the computational cost for applying the transformations. In a global mapping approach, at each step we only have to apply a transformation to the latest acquired cloud (see Equation 6.1), while in the robocentric approach we align the entire map to the cloud (see Equation 6.4). If the map is significantly bigger than the cloud, this could cause a computational overhead. In our case, since the map is bounded and filtered, the difference is negligible.

Although more sophisticated mapping representations are available, the combination of point cloud mapping and on-demand local heightmaps has proven to be effective for a static

and dynamic locomotion application. In these situations, simplicity and speed are of utmost priority, because the robot stability depends on promptly delivered information. When the execution of more complicated motions in cluttered environments is required, (*e.g.,* chimney climbs [43], crawling inside tunnels, *etc.*), these mapping solutions may be insufficient. Different representations are also required for long term navigation. As we discussed in Chapter 5 for pose estimation, for mapping we also devise the combination of multiple solutions as the most promising technique. For example, in a long term mission, a hierarchy of maps could provide an appropriate solution for the many involved problems: topological maps for global path planning; point cloud maps for foothold planning; heightmaps for dynamic and reactive locomotion; OctoMaps or meshes for planning in constrained environments.

6.5 Conclusion

In this chapter, we have described different mapping approaches and their applications to static and dynamic locomotion. In particular, we have explained the concept of robocentric mapping and hightlighted its benefits for locomotion when compared to global mapping. We have also presented two algorithms for terrain modeling: point cloud mapping and local elevation mapping. These two mapping algorithms have been successfully used for obstacle traversal in both static and dynamic gaits. In particular, for the dynamic gaits we have introduced a novel approach to terrain classification in-the-loop based on logistic regression which enables the robot to reactively negotiate obstacles in real-time while trotting. This work was the first showing the potential of terrain classification in-the-loop during dynamic legged motions. The work has been published on [9] as second author.

Future research will head towards the inclusion of more information in the maps (*e.g.,* uncertainties) and the incorporation of additional mapping algorithms for long term navigation.

Chapter 7

Conclusion and Future Work

The final goal of the HyQ project is to create a vehicle for extreme terrain, able to autonomously negotiate obstacles and traverse them, similar to what legged animals have been doing for millions of years. Since the construction of the first robot prototype in 2010, the HyQ project has significantly moved forward the research in legged locomotion. In less than five years, the team of the Dynamic Legged System Lab was able to publicly demonstrate a variety of gaits and locomotion skills, including[1]: planned walk on uneven terrain [144], reactive locomotion over rough terrain [8], balancing under disturbances, highly dynamic motions (with aerial phase), step reflex [42], chimney climbing [43]. These demonstrations have shown the true potential of legged locomotion, when the terrain is too difficult for wheels and tracked vehicles.

The inherent complexity in achieving this goal resides in the close relationship between perception and locomotion. Every millisecond, the robot has to know the status of its joints, its location in the world, its base velocity, and what the terrain looks like. At the same time it has to process all of this data, and generate the appropriate actuator commands to propel itself, without falling, slipping or getting stuck. These operations have to be executed on multiple scales, from low level joint position and torque tracking, to foot trajectory generation and foothold selection, and even to global path planning and navigation over long distances.

Given the amount of complexity involved, most of the research effort has been focused on control theory and locomotion aspects of the problem. The role of perception and reasoning was instead limited to the minimum level required to complete a specific task. Often, some of the perception capabilities were replaced by surrogates, such as motion capture systems for localization. For this reason, there are only a few examples of field-capable autonomous legged robots, while the rest are still confined to laboratories and test facilities.

The objective of this dissertation was to introduce the essential algorithms and tools required to achieve true autonomy in terms of perception, and to reduce the gap between perception and locomotion. After three years of a PhD, the HyQ robot has attained unprecedented levels of perception-aware locomotion. In particular, we have achieved the following milestones:

[1] A summary video of these capabilities is available at: http://www.youtube.com/watch?v=ENHvCGrnr2g

- integration of an EKF-based state estimator, which fuses an inertial process model and a novel LO algorithm, to achieve accurate and smooth velocity estimates. These are fundamental for attitude control and disturbance rejection, especially when performing dynamic motions, such as trotting. Experiments with in-the-loop state estimation have shown significant improvement in the trunk position control when trotting on uneven surfaces;
- fusion of VO- and LiDAR-based localization algorithms within the above mentioned filter to achieve very accurate pose estimation, even in adverse conditions, including: scarce illumination, motion blur, uneven and rough terrain, sharp motion. The level of accuracy of the pose estimate, as well as its stability, is demonstrated by the successful execution of multiple in-the-loop experiments. These show a seamless base pose tracking of a practically drift-free estimated trajectory;
- development of a lightweight, robocentric point cloud mapping algorithm, with on-demand local elevation mapping capabilities;
- tight integration of the mapping algorithm within the control loop, to modulate the trotting behavior and to perform reactive vision-in-the-loop obstacle negotiation.

Future directions of research include:

Additional tests. We are interested in concurrently executing all the elements explained above, on the robot. In particular, to decouple the problems of state estimation and mapping, the state estimator has been tested within the control loop, but with mapping not actively used. On the other hand, the mapping algorithm was actively used in the control loop, but with an external source of localization;

Terrain assessment and classification. We would like to continue the research in terrain classification by introducing more sophisticated techniques to modify the locomotion behavior, depending on the terrain characteristics;

Advanced SLAM techniques. Future long-term missions in unknown environments will require large scale mapping. In this context, the integration of typical SLAM techniques [95], like loop-closure detection and pose graph optimization [73], will be investigated.

Appendices

Appendix A

Datasets

All the algorithms presented in this dissertation have been tested extensively on HyQ. Where possible, the data used for performance assessment have been collected and organized in a dataset. In this appendix we describe the content and the characteristics of three datasets, collected between the summer of 2015 and the end of 2016. Appendix A.1 describes the Dataset 1, used for the performance assessment of the LiDAR-based localization modules described in Sections 5.3 and 5.4. Appendix A.2 describes the Dataset 2, mainly used to test the state estimation framework with inertial and LO inputs described in Chapter 4; Appendix A.3 describes the Dataset 3, used to test the multi-sensor approach of Section 5.5 in challenging conditions for the perception. All the datasets have been recorded in LCM format [63].

A.1 Dataset 1

This dataset was recorded in mid 2015 with the robot in Configuration A (see Section 3.5.1). It consists of six logs (three with trotting gait and three crawling gait) for a total of $\approx 8\,\mathrm{min}$. Each datalog includes the following sensor measurements/estimations and additional information:

- joint states at $250\,\mathrm{Hz}$ (position $\mathbf{q} \in \mathbb{R}^{12}$, velocity $\dot{\mathbf{q}} \in \mathbb{R}^{12}$ and effort $\boldsymbol{\tau} \in \mathbb{R}^{12}$);
- IMU proper acceleration ${}_i\mathbf{a}_i \in \mathbb{R}^3$ and angular velocity ${}_i\boldsymbol{\omega}_i \in \mathbb{R}^3$, measured from the Inertial Measurement Unit;
- Velodyne HDL-32E laser scans at $10\,\mathrm{Hz}$;
- Hokuyo URG-04LX laser scans at $10\,\mathrm{Hz}$;
- ASUS Xtion Pro point clouds at $30\,\mathrm{Hz}$;
- Pan and Tilt Unit (PTU) angle positions (pan and tilt) at $50\,\mathrm{Hz}$;
- robot base position ${}_w\mathbf{x}_b$ at $100\,\mathrm{Hz}$, in the world frame, from Vicon motion capture system, used as ground truth;

Name	Gait	Duration [s]	Distance [m]
d1-trot-01	trot	78	4.02
d1-trot-02	trot	91	4.55
d1-trot-03	trot	82	4.98
d1-crawl-04	crawl	39	0.87
d1-crawl-05	crawl	83	1.25
d1-crawl-06	crawl	89	1.49

Table A.1: Summary of Dataset 1.

- robot kinematic tree model.

A summary of the datalogs is provided in Table A.1: it contains the datalog ID, the type of gait (trot or walk), total duration, and total distance traveled, which has been computed by integration of the absolute velocity estimated from Vicon position measurements.

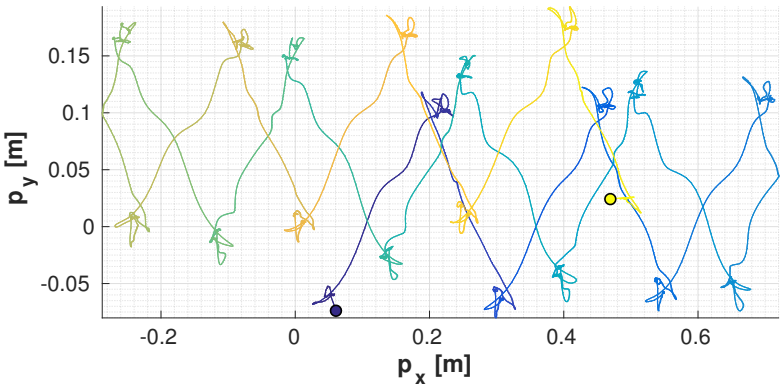
A.2 Dataset 2

The Dataset 2 has been recorded from winter 2015 to mid 2016 for the experimental tests on the proprioceptive state estimator described in Chapter 4. The datalogs include the following signals:

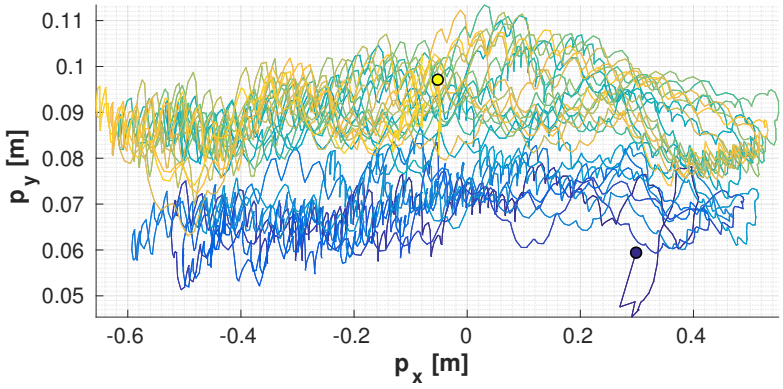
- joint states at 250 Hz (position $\mathbf{q} \in \mathbb{R}^{12}$, velocity $\dot{\mathbf{q}} \in \mathbb{R}^{12}$ and effort $\boldsymbol{\tau} \in \mathbb{R}^{12}$);
- Microstrain 3DM-GX3-25 signals at 250 Hz (proper acceleration ${}_i\mathbf{a}_i \in \mathbb{R}^3$ and angular velocity ${}_i\boldsymbol{\omega}_i \in \mathbb{R}^3$);
- Hokuyo URG-04LX laser scans at 10 Hz;
- robot base position ${}_w\mathbf{x}_b$ at 100 Hz (from Vicon motion capture system, used as ground truth);
- robot kinematic tree model.

The dataset is summarized in Table A.2: it consists of seven runs, three of a trotting gait and four of a crawling gait, for a total duration of 62 min. The total distance traveled was computed by path integral of the robot trajectory from Vicon position measurements at 100 Hz. The velocity signals are computed by numerical differentiation and de-noised through a delay-compensated second order Savitzky-Golay filter [126], which was preferred over a moving average for its smaller signal distortion.

Due to the limited size of our motion capture space, these gaits were performed by repeating forward-backward motions within a $2.5 \times 1.2\,\mathrm{m}^2$ area. Figures A.1a and A.1b depict a typical trajectory (projected onto the xy-plane) of a crawl and a trot run, respectively.



(a) Crawl trajectory



(b) Trot trajectory

Figure A.1: Typical crawl and trot base trajectories, projected on the xy-plane. Color changes from dark blue to light yellow indicate the evolution of time. Starting and ending positions are indicated by a blue and a yellow dot, respectively.

Name	Gait	Duration [s]	Distance [m]
d2-trot-01	trot	606	38.55
d2-trot-02	trot	608	40.85
d2-trot-03	trot	609	48.96
d2-crawl-04	crawl	395	8.05
d2-crawl-05	crawl	345	6.94
d2-crawl-06	crawl	600	10.31
d2-crawl-07	crawl	600	10.89

Table A.2: Summary of Dataset 2.

Name	Gait	Duration [s]	Distance [m]
d3-trot-01	trot	313	∼18
d3-trot-02	trot	330	∼20
d3-trot-03	trot	469	∼32
d3-crawl-04	crawl	869	∼17
d3-crawl-05	crawl	675	∼13.5

Table A.3: Summary of Dataset 3. The distance traveled is approximated, since the ground truth was not available.

A.3 Dataset 3

The Dataset 3 has been recorded in winter 2016 for the experimental tests on the multi-sensor state estimator described in Section 5.5. In contrast with the previous datasets, the datalogs contain data collected with the robot in Configuration B. The signals from the 3DM-GX4-25 and the KVH 1775 have been synchronized with the joint states in real-time (through dedicated EtherCAT boards) for signal comparison purposes. The dataset includes the following signals:

- joint states at 1 kHz (position $\mathbf{q} \in \mathbb{R}^{12}$, velocity $\dot{\mathbf{q}} \in \mathbb{R}^{12}$ and effort $\boldsymbol{\tau} \in \mathbb{R}^{12}$);
- Microstrain 3DM-GX3-25 and KVH 1775 IMU signals at 1 kHz (proper acceleration ${}_i\mathbf{a}_i \in \mathbb{R}^3$ and angular velocity ${}_i\boldsymbol{\omega}_i \in \mathbb{R}^3$) measured from the Inertial Measurement Unit (Microstrain 3DM-GX3-25);
- Stereo and depth images at 10 Hz (from Multisense SL)
- Hokuyo URG-04LX at 10 Hz laser scans;
- Hokuyo UTM-30LX at 40 Hz (from Multisense SL);
- robot and Multisense SL kinematic tree model.

The dataset is summarized in Table A.3: it consists of five runs, three of a trotting gait and two of a crawling gait, for a total duration of 44 min. All the datalogs (except *d3-trot-03*) have been recorded on an extended test area of $20 \times 5\,\mathrm{m}^2$ (Figure A.4), with different types of obstacles, ramps and rocks (see Figures A.3b, A.4 and Section 5.5). The environmental conditions were challenging for the perception system: because the experiments were taken at night, the image is underexposed on the sides and overexposed at the center (due to flashlights). Furthermore, the protection frame for the camera occludes part of the scene (see Figure A.3)

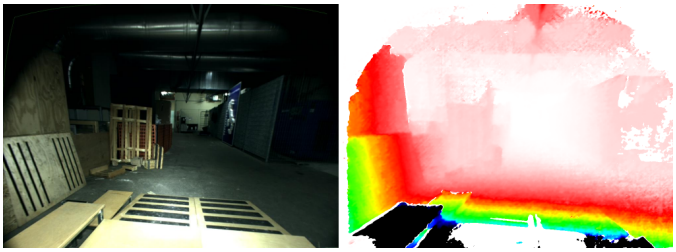
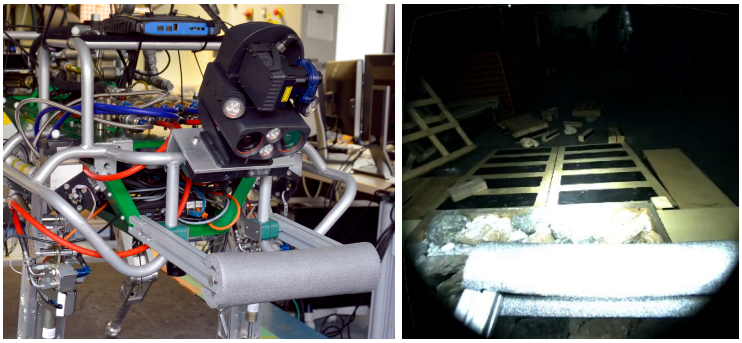


Figure A.2: Cropped camera and depth images from Multisense SL. Note the challenging lighting conditions and structure of the test arena.



(a) Protection frame **(b)** Camera image

Figure A.3: Detail of the Multisense SL protection frame and the corresponding occlusion on the image.

Figure A.4: Test area for Dataset 3. The area in picture is displayed without obstacles. During the individual runs, ramps and rocks have been added (see Figure A.2).

Appendix B

Gaussian Filters

Gaussian Filters model the state to be observed, $\boldsymbol{x}$, as a random variable whose density function is defined by a multi-variate Gaussian distribution. The constitutive parameters of this distribution are the mean $\boldsymbol{\mu}$ and the covariance matrix Σ:

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{2\pi|\Sigma|}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right] \tag{B.1}$$

In the following sections, we describe three widely used types of Gaussian filters: the Kalman Filter, the Extended Kalman Filter, and the Unscented Kalman Filter [140]. We conclude the appendix with the Particle Filter and its Gaussian approximation.

B.1 Kalman Filter

The Kalman Filter [72] is an optimal recursive state observer for linear Gaussian systems. Let $\boldsymbol{x}_k \in \mathbb{R}^{n\times 1}$ be the state vector to be estimated, $\boldsymbol{u}_k \in \mathbb{R}^{n\times 1}$ a control action, and $\boldsymbol{z}_k \in \mathbb{R}^{m\times 1}$ a measurement coming from a sensor, all evaluated at time step $k \in \mathbb{N}$. Their relationship, assumed to be linear, is stated as follows:

$$\boldsymbol{x}_k(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) = A_k\boldsymbol{x}_{k-1} + B_k\boldsymbol{u}_k + \boldsymbol{v}_k \tag{B.2}$$

$$\boldsymbol{z}_k(\boldsymbol{x}_k) = C_k\boldsymbol{x}_k + \boldsymbol{w}_k \tag{B.3}$$

where zero mean Gaussian additive terms $p(\boldsymbol{v}_k) = \mathcal{N}(0, R_k)$ and $p(\boldsymbol{w}_k) = \mathcal{N}(0, Q_k)$ are defined as *process noise* and *measurement noise*, respectively. In (B.2) the process is assumed to be Markovian. Hence, the current state $\boldsymbol{x}_k$ depends only on the previous state $\boldsymbol{x}_{k-1}$ (which incorporates all the information about the past states $\boldsymbol{x}_{0:k-2}$ and measurements $\boldsymbol{z}_{0:k-1}$) and the current control action $\boldsymbol{u}_k$. The parameters $A_k \in \mathbb{R}^{n\times n}$, $B_k \in \mathbb{R}^{n\times n}$, and $C_k \in \mathbb{R}^{m\times n}$ depend on the timestep k only.

Since the state is modeled as Gaussian, at each iteration k the ouput is a mean $\boldsymbol{\mu} = \hat{\boldsymbol{x}}_k$ and

a covariance matrix $\Sigma = P_k$.

Prediction

$$\hat{\boldsymbol{x}}_k^- = A_k \hat{\boldsymbol{x}}_{k-1} + B_k \boldsymbol{u}_k \tag{B.4}$$

$$P_k^- = A_k P_{k-1} A_k^{\mathrm{T}} + R_k \tag{B.5}$$

Update

$$K_k = P_k^- C_k^{\mathrm{T}} (C_k P_k^- C_k^{\mathrm{T}} + Q_k)^{-1} \tag{B.6}$$

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + K_k(\boldsymbol{z}_k - C_k \hat{\boldsymbol{x}}_k^-) \tag{B.7}$$

$$P_k = (I - K_k C_k) P_k^- \tag{B.8}$$

During the prediction step, in (B.4) the prior state $\boldsymbol{x}_k^-$ is predicted using the process model equation B.2, while the predicted covariance P_k^- is computed in (B.5) by propagation of the previous covariance and addition of the process noise covariance.

When a new measurement $\boldsymbol{z}_k$ is available, the priors $\boldsymbol{x}_k^-$ and P_k^- are updated: (B.6) computes the Kalman gain K_k, a quantity which expresses how much weight the filter gives to the new information carried by the measurement. A gain of $K_k = 0$ will just ignore the measurement, whereas a gain of $K_k = 1$ will use the measurement as the only source to compute the state, as shown by the second term of (B.7). Finally, in (B.8) the covariance is updated by means of the Kalman gain, similar to what is done for the state.

The Kalman filter relies on the assumptions that the state, noise and measurements are Gaussian, and the system is linear. These assumptions are unrealistic in many applications – especially for the linearity assumption. The filters described in the following sections provide approximation techniques to relax the linearity assumption.

B.2 Extended Kalman Filter

The EKF was originally developed to accommodate for the nonlinearity of the transition and measurement functions [142]:

$$\boldsymbol{x}_k(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) = g(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{v}_k) \tag{B.9}$$

$$\boldsymbol{z}_k(\boldsymbol{x}_t) = h(\boldsymbol{x}_k, \boldsymbol{w}_k) \tag{B.10}$$

This is achieved by linearization of $g(\cdot)$ and $h(\cdot)$ through a first order Taylor series expansion, where they are replaced as follows. We define the followining quantities:

$$G_k = \left.\frac{\partial g(\boldsymbol{x}, \boldsymbol{u}_k)}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_{k-1}} \tag{B.11}$$

$$H_k = \left.\frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_k^-} \tag{B.12}$$

$$g(\boldsymbol{x}, \boldsymbol{u}_k) \approx g(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_k) + G_k(\boldsymbol{x} - \hat{\boldsymbol{x}}_{k-1}) \tag{B.13}$$

$$h(\boldsymbol{x}) \approx h(\hat{\boldsymbol{x}}_k^-) + H_k(\boldsymbol{x} - \hat{\boldsymbol{x}}_k^-) \tag{B.14}$$

After being linearized, the operations for the filter are similar to the linear version:

Prediction

$$\hat{\boldsymbol{x}}_k^- = g(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_k) \tag{B.15}$$

$$P_k^- = G_k P_{k-1} G_k^{\mathrm{T}} + V_k R_k V_k^{\mathrm{T}} \tag{B.16}$$

Update

$$K_k = P_k^- H_k^{\mathrm{T}} (H_k P_k^- H_k^{\mathrm{T}} + Q_k)^{-1} \tag{B.17}$$

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + K_k(\boldsymbol{z}_k - H_k \hat{\boldsymbol{x}}_k^-) \tag{B.18}$$

$$P_k = (I - K_k H_k) P_k^- \tag{B.19}$$

B.3 Unscented Kalman Filter

An alternative solution to recover approximated mean and covariance of a distribution after nonlinear transformation is provided by the UKF. Originally proposed by Julier and Uhlmann [70], this variant of the Kalman filter is based on the intuition that, instead of approximating a known non-linear function in order to recover mean and variance of the fitting Gaussian, these parameters can be estimated by transforming a finite set of points from the initial distribution and seeing how the shape of the original Gaussian has changed. With $2n + 1$ points, the estimated mean and variance are closer to the true distribution than the ones estimated with the EKF, with the additional benefit that no derivatives are required. Let us consider a sequence of $2n + 1$ "sigma" points computed as follows:

$$\mathcal{X}^{[0]} = \mu \tag{B.20}$$

$$\mathcal{X}^{[i]} = \mu + \sqrt{(n + \lambda)\Sigma} \qquad i = 1, \ldots, n \tag{B.21}$$

$$\mathcal{X}^{[i]} = \mu - \sqrt{(n + \lambda)\Sigma} \qquad i = n + 1, \ldots, 2n \tag{B.22}$$

where $\lambda = \alpha^2(n + \kappa) - n$, and α, κ are scaling parameters that determine the distance of the sigma points from the mean. The sigma points are passed to the function g, to reconstruct how

it is distorting the original mean and covariance:

$$\mu' = \sum_{i=0}^{2n} w_m^{[i]} g(\mathcal{X}^{[i]}) = \sum_{i=0}^{2n} w_m^{[i]} \mathcal{Y}^{[i]} \tag{B.23}$$

$$\Sigma' = \sum_{i=0}^{2n} w_c^{[i]} (\mathcal{Y}^{[i]} - \mu')(\mathcal{Y}^{[i]} - \mu')^{\mathrm{T}} \tag{B.24}$$

By combining (B.23) and (B.24) into the Kalman filter framework, we obtain (for simplicity, the one-dimensional state x is used):

Prediction (B.25)

$$\mathcal{X}_{k-1} = \left[\hat{x}_{k-1}, \quad \hat{x}_{k-1} + \sqrt{(n+\lambda)\Sigma}, \quad \hat{x}_{k-1} - \sqrt{(n+\lambda)\Sigma}\right] \tag{B.26}$$

$$\bar{\mathcal{X}}_k = g(\mathcal{X}_{k-1}, u_k) \tag{B.27}$$

$$\hat{x}_k^- = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{X}}_k \tag{B.28}$$

$$P_k^- = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{X}}^{[i]} - \bar{\mu}_k)(\bar{\mathcal{X}}^{[i]} - \bar{\mu}_k)^{\mathrm{T}} + R_k \tag{B.29}$$

Update (B.30)

$$\mathcal{Z}_k = h(\bar{\mathcal{X}}_k) \tag{B.31}$$

$$\hat{z}_k = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{Z}}_k^{[i]} \tag{B.32}$$

$$S_k = \sum_{i=0}^{2n} w_c^{[i]} (\mathcal{Z}_k^{[i]} - \hat{z}_k)(\mathcal{Z}_k^{[i]} - \hat{z}_k)^{\mathrm{T}} + Q_k \tag{B.33}$$

$$\bar{\Sigma}_k^{x,y} = \sum_{i=0}^{2n} w_c^{[i]} (\mathcal{X}_k^{[i]} - \hat{\mu}_k)(\mathcal{Z}_k^{[i]} - \hat{z}_k)^{\mathrm{T}} \tag{B.34}$$

$$K_k = \bar{\Sigma}_k^{x,y} S_k^{-1} \tag{B.35}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - \hat{z}_k) \tag{B.36}$$

$$P_k = P_k^- - K_k S_k K_k^{\mathrm{T}} \tag{B.37}$$

B.4 Gaussian Particle Filter

The *particle filter* models the posterior distribution with a finite set of samples called *particles*, each one representing a different hypothesis of the real state [141]. In contrast with the Gaussian distribution, this representation is non-parametric, and provides more freedom in terms of possible shapes of the posterior distribution (*e.g.,* multimodal distributions).

The basic intuition behind this algorithm is to approximate the posterior distribution with a discrete set of M sample points $x^{[0]}, \ldots, x^{[M]} \in \mathcal{X}$ drawn from a set of hypotheses $\bar{\mathcal{X}}$ which approximate the prior distribution. The full algorithm, which involves the integration of

measurements is as follows:

1: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2: **for** $m = 1$ **to** $m = M$ **do**
3: sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$
4: $w_t^{[m]} = p(z_t|x_t^{[m]})$
5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + (x_t^{[m]}, w_t^{[m]})$
6: **end for**
7: **for** $m = 1$ **to** $m = M$ **do**
8: draw i with probability $\propto w_t^{[m]}$
9: add $x_t^{[i]}$ to $\mathcal{X}_t$
10: **end for**
11: return $\mathcal{X}_t$

At line 3, the algorithm constructs its temporary set of hypotheses $x_t^{[m]}$ by drawing from the prior distribution, $p(x_t|u_t, x_{t-1}^{[m]})$. Then, from the measurement update $p(z_t|x_t^{[m]})$, it updates the importance weight $w_t^{[m]}$. The importance weight incorporates the information provided by the measurement: the hypotheses which are consistent with the measurement are assigned to a bigger weight. The posterior is computed from lines 7–9: each new particle is the result of an drawing with replacement from the prior distribution, proportional to the weights (line 8). The extraction using the weight is the crucial part of the algorithm, because it incorporates the measurement into the estimation. If a particle is assigned with a small weight, it will have a small chance to get collected for the posterior. This allows to gather the particles in the region where the posterior has high probability.

The Gaussian particle filter is a special case of particle filter where the approximated distribution is Gaussian. The approximated Gaussian distribution is computed from the weighted mean and covariance of the population of particles [77].

Appendix C

Performance Metrics

C.1 Drift per Distance Traveled

The performance metric used to assess position accuracy throughout this dissertation is the Drift per Distance Traveled (DDT), which for display convenience is expressed $\mathrm{cm/m}$ or $\%$. Given N samples of the true position $\mathbf{x} = [x, y, z]^{\mathrm{T}}$, and the estimated position $\hat{\mathbf{x}} = [\hat{x}, \hat{y}, \hat{z}]^{\mathrm{T}}$, the DDT for the x-axis is defined as follows:

$$\mathrm{DDT}_x = 100 \frac{\frac{1}{N}\sum_{k=1}^{N} |x_k - \hat{x}_k|}{\sum_{k=1}^{N-1} |x_{k+1} - x_k|} = 100 \frac{\bar{e}_x}{\sum_{k=1}^{N-1} \Delta_x} \tag{C.1}$$

The norm of the DDT for the three axes gives the total DDT, as:

$$\mathrm{DDT} = 100 \frac{\sqrt{\bar{e}_x^2 + \bar{e}_y^2 + \bar{e}_z^2}}{\sum_{k=1}^{N-1} \sqrt{\Delta_x^2 + \Delta_y^2 + \Delta_z^2}} \tag{C.2}$$

where the numerator expresses the norm of average of the absolute difference between estimated position $\hat{\mathbf{x}}$ and the ground truth $\mathbf{x}$, while the denominator express the discrete path integral of the position computed from the ground truth (*i.e.,* the total distance traveled). The metric expresses the average distance from the real position, normalized by the total distance traveled. Given the sub-millimetric accuracy of the motion capture system, the denominator of (C.1) may assume larger values than expected, since the integral is including all the small "ripples" in the trajectory (*cf.* Figure A.1b). However, the estimator usually operates at higher frequencies than the motion capture system. Therefore, the estimator captures the same level of detail and compensates for the high frequency components of the trajectory.

C.2 Root Mean Square Error

The Root Mean Squared Error is a commonly used metric for comparing the performance of velocity estimators. Given N samples of ground truth velocity $\dot{\mathbf{x}}_k$ and estimated velocity $\hat{\dot{\mathbf{x}}}_k$, the RMSE is defined as the root of the mean of the absolute quadratic error $|\dot{\mathbf{x}} - \hat{\dot{\mathbf{x}}}|$:

$$\text{RMSE} = \sqrt{\frac{\sum_{k=1}^{N} |\dot{\mathbf{x}}_k - \hat{\dot{\mathbf{x}}}_k|^2}{N}} \tag{C.3}$$

Bibliography

[1] 2015 in review: Winning year for robots and for kvh. www.kvhmobileworld.kvh.com.

[2] Ethercat technology group. https://www.ethercat.org/.

[3] Ntp: The network time protocol. http://www.ntp.org/.

[4] Zakaria Ahmad. xPC Target Real-Time Environment for The MIT Cheetah Robot. Technical report, Georgia Institute of Technology, August 2013.

[5] A. Bachrach, S. Prentice, R. He, and N. Roy. RANGE - Robust Autonomous Navigation in GPS-denied Environments. *Journal of Field Robotics*, 28(5):644–666, 9 2011.

[6] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (SLAM): part II. *Robotics Automation Magazine, IEEE*, 13(3):108–117, 9 2006.

[7] Max Bajracharya, Jeremy Ma, Matt Malchano, Alex Perkins, Alfred A. Rizzi, and Larry Matthies. High Fidelity Day/Night Stereo Mapping with Vegetation and Negative Obstacle Detection for Vision-in-the-Loop Walking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3663–3670, November 2013.

[8] Victor Barasuol, Jonas Buchli, Claudio Semini, Marco Frigerio, Edson Roberto De Pieri, and Darwin G. Caldwell. A Reactive Controller Framework for Quadrupedal Locomotion on Challenging Terrain. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2554–2561, May 2013.

[9] Victor Barasuol, Marco Camurri, Stephane Bazeille, Darwin Caldwell, and Claudio Semini. Reactive trotting with foot placement corrections through visual pattern classification. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 10 2015.

[10] Stéphane Bazeille, Victor Barasuol, Michele Focchi, Ioannis Havoutis, Marco Frigerio, Jonas Buchli, Darwin G. Caldwell, and Claudio Semini. Quadruped robot trotting over irregular terrain assisted by stereo-vision. *Intelligent Service Robotics*, 7(2):67–77, 2014.

[11] Stéphane Bazeille, Marco Camurri, Jesus Ortiz, Ioannis Havoutis, Darwin G. Caldwell, and Claudio Semini. Terrain mapping with a pan and tilt stereo camera for locomotion on a quadruped robot. In *ICRA14 Workshop on Modelling, Estimation, Perception and Control of All Terrain Mobile Robots (WMEPC14)*, 2014.

[12] Stéphane Bazeille, Jesus Ortiz, Francesco Rovida, Marco Camurri, Anis Meguenani, Darwin G. Caldwell, and Claudio Semini. Active camera stabilization to enhance the vision of agile legged robots. *Robotica*, pages 1–19, November 2015.

[13] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, February 1992.

[14] A. Bethencourt and L. Jaulin. 3d reconstruction using interval methods on the kinect device coupled with an imu. *International Journal of Advanced Robotic Systems*, 10(93):1–10, 2013.

[15] Peter Biber and Wolfgang Straßer. The Normal Distributions Transform: A New Approach to Laser Scan Matching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2743–2748, October 2003.

[16] M. Bloesch, C. Gehring, P. Fankhauser, M. Hutter, M.A. Hoepflinger, and R. Siegwart. State estimation for legged robots on unstable and slippery terrain. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6058–6064, November 2013.

[17] Michael Bloesch, Marco Hutter, Mark Hoepflinger, Stefan Leutenegger, Christian Gehring, C. David Remy, and Roland Siegwart. State estimation for legged robots - consistent fusion of leg kinematics and IMU. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[18] Michael Bosse, Robert Zlot, and P. Flick. Zebedee: Design of a spring-mounted 3-d range sensor with application to mobile mapping. *Robotics, IEEE Transactions on*, 28(5):1104–1119, October 2012.

[19] Thierry Bouwmans. Traditional Approaches in Background Modeling for Static Cameras. In *Background Modeling and Foreground Detection for Video Surveillance*, pages 1–54. Chapman and Hall/CRC, July 2014.

[20] A. Bowling. Impact forces and mobility in legged robot locomotion. In *IEEE/ASME international conference on Advanced intelligent mechatronics*, pages 1–8, September 2007.

[21] David M. Bradley, Jonathan K. Chang, David Silver, Matthew Powers, Herman Herman, Peter Rander, and Anthony Stentz. Scene understanding for a high-mobility walking robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1144–1151, 9 2015.

[22] A. Bry, A. Bachrach, and N. Roy. State estimation for aggressive flight in GPS-denied environments using onboard sensing. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, May 2012.

[23] Wolfram Burgard and Martial Hebert. *World Modeling*, pages 853–869. Springer Berlin Heidelberg, 2008.

[24] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, December 2016.

[25] Gabriele Camellini, Mirko Felisa, Paolo Medici, Paolo Zani, Francesco Gregoretti, Claudio Passerone, and Roberto Passerone. 3dv — an embedded, dense stereovision-based depth mapping system. In *Procs. IEEE Intelligent Vehicles Symposium 2014*, pages 1435–1440, Dearbon, MI, USA, 6 2014.

[26] Marco Camurri, Stéphane Bazeille, Darwin G. Caldwell, and Claudio Semini. Real-time depth and inertial fusion for local SLAM on dynamic legged robots. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 259–264, September 2015.

[27] Marco Camurri, Maurice Fallon, Stéphane Bazeille, Andreea Radulescu, Victor Barasuol, Darwin G. Caldwell, and Claudio Semini. Probabilistic contact estimation and impact detection for state estimation of quadruped robots. *IEEE Robotics and Automation Letters*, pages 1023–1030, April 2017.

[28] J. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, November 1986.

[29] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145 – 155, 1992.

[30] Annett Chilian, Heiko Hirschmüller, and Martin Görner. Multisensor data fusion for robust pose estimation of a six-legged walking robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2497–2504. IEEE, 2011.

[31] S. Chitta, P. Vemaza, R. Geykhman, and D. D. Lee. Proprioceptive localization for a quadrupedal robot on known terrain. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4582–4587, April 2007.

[32] J. Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors, October 2006. Stanford University.

[33] H. Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *Robotics Automation Magazine, IEEE*, 13(2):99–110, 6 2006.

[34] N. El-Sheimy, H. Hou, and X. Niu. Analysis and modeling of inertial sensors using allan variance. *IEEE Transactions on Instrumentation and Measurement*, 57(1):140–149, January 2008.

[35] Jakob Engel, Jörg Stückler, and Daniel Cremers. Large-scale direct slam with stereo cameras. In *International Conference on Intelligent Robots and Systems (IROS)*, September 2015.

[36] Maurice Fallon, Pat Marion, Robin Deits, Thomas Whelan, Matthew Antone, John McDonald, and Russ Tedrake. Continuous Humanoid Locomotion over Uneven Terrain using Stereo Fusion. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 881–888, November 2015.

[37] M.F. Fallon, M. Antone, N. Roy, and S. Teller. Drift-free humanoid state estimation fusing kinematic, inertial and lidar sensing. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 112–119, November 2014.

[38] P Fankhauser, M Bloesch, C Gehring, M Hutter, and Roland Yves Siegwart. Robot-centric elevation mapping with uncertainty estimates. *Proceedings of CLAWAR*, 2014.

[39] Roy Featherstone. *Robot Dynamics Algorithms*. Kluwer Academic Publishers, 1987.

[40] Roy Featherstone and David E. Orin. *Dynamics*, pages 35–65. Springer Berlin Heidelberg, 2008.

[41] P. Filitchkin and K. Byl. Feature-based terrain classification for littledog. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1387–1392, 10 2012.

[42] Michele Focchi, Victor Barasuol, Ioannis Havoutis, Jonas Buchli, Claudio Semini, and Darwin G. Caldwell. Local Reflex Generation for Obstacle Negotiation in Quadrupedal Locomotion. In *International Conference on Climbing and Walking Robots (CLAWAR)*, 2013.

[43] Michele Focchi, Andrea del Prete, Ioannis Havoutis, Roy Featherstone, Darwin G. Caldwell, and Claudio Semini. High-slope terrain locomotion for torque-controlled quadruped robots. *Autonomous Robots*, 41(1):259–272, 2017.

[44] F. Fraundorfer and D. Scaramuzza. Visual odometry : Part ii: Matching, robustness, optimization, and applications. *IEEE Robotics Automation Magazine*, 19(2):78–90, June 2012.

[45] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014.

[46] B. Gaßmann, K. U. Scholl, and K. Berns. Locomotion of LAURON III in rough terrain. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics.*, volume 2, pages 959–964, July 2001.

[47] B. Gassmann, F. Zacharias, J. M. Zollner, and R. Dillmann. Localization of walking robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1471–1476, April 2005.

[48] Martin Görner and Annett Stelzer. A leg proprioception based 6 dof odometry for statically stable walking robots. *Autonomous Robots*, 34(4):311–326, 2013.

[49] Martin Görner, Thomas Wimbock, Andreas Baumann, Matthias Fuchs, Thomas Bahls, Markus Grebenstein, Christoph Borst, Jörg Butterfass, and Gerd Hirzinger. The dlr-crawler: A testbed for actively compliant hexapod walking based on the fingers of dlr-hand ii. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1525–1531, Sep 2008.

[50] M. Grewal and A. Andrews. How good is your gyro [ask the experts]. *IEEE Control Systems*, 30(1):12–86, February 2010.

[51] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, winter 2010.

[52] O. Gür and U. Saranli. Model-based proprioceptive state estimation for spring-mass running. *Robotics: Science and Systems VII*, pages 105–112, 2012.

[53] S. Haddadin, A. Albu-Schaffer, A. De Luca, and G. Hirzinger. Collision detection and reaction: A contribution to safe physical human-robot interaction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3356–3363, September 2008.

[54] H. Haggag, M. Hossny, D. Filippidis, D. Creighton, S. Nahavandi, and V. Puri. Measuring depth accuracy in rgbd cameras. In *7th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pages 1–7, December 2013.

[55] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*, chapter 4. Springer, 2 edition, 2009.

[56] Ioannis Havoutis, Jesus Ortiz, Stephane. Bazeille, Victor Barasuol, Claudio Semini, and Darwin G. Caldwell. Onboard Perception-Based Trotting and Crawling with the Hydraulic Quadruped Robot (HyQ). In *IEEE/RSJ International Conference on Intelligent Robotsand Systems (IROS)*, 2013.

[57] Andreas Hermann, Florian Drews, Joerg Bauer, Sebastian Klemm, Arne Roennau, and Ruediger Dillmann. Unified GPU Voxel Collision Detection for Mobile Manipulation Planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4154–4160, Sep 2014.

[58] H. Hirschmuller, P.R. Innocent, and J.M. Garibaldi. Fast, unconstrained camera motion estimation from stereo without tracking and robust statistics. *7th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2:1099–1104, December 2002.

[59] M. A. Hoepflinger, M. Hutter, C. Gehring, M. Bloesch, and R. Siegwart. Unsupervised identification and prediction of foothold robustness. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3293–3298, May 2013.

[60] Armin Hornung, Stefan Oßwald, Daniel Maier, and Maren Bennewitz. Monte Carlo Localization for Humanoid Robot Navigation in Complex Indoor Environments. *International Journal of Humanoid Robotics*, 11(02):1441002, 2014.

[61] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[62] Andrew Howard. Real-time stereo visual odometry for autonomous ground vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3946–3952. IEEE, Sep 2008.

[63] A. S. Huang, E. Olson, and D. C. Moore. Lcm: Lightweight communications and marshalling. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4057–4062, October 2010.

[64] Albert S. Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. *Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera*, pages 235–252. Springer International Publishing, 2017.

[65] Marco Hutter, Christian Gehring, Michael Bloesch, Mark A Hoepflinger, C David Remy, and Roland Siegwart. Starleth: A compliant quadrupedal robot for fast, efficient, and versatile locomotion. In *International Conference on Climbing and Walking Robot (CLAWAR)*, July 2012.

[66] Marco Hutter, Christian Gehring, Dominic Jud, Andreas Lauber, C. Dario Bellicoso, Vassilios Tsounis, Jemin Hwangbo, Karen Bodie, Peter Fankhauser, Michael Bloesch, Remo Diethelm, Samuel Bachmann, Amir Melzer, and Mark Hoepflinger. ANYmal - A Highly Mobile and Dynamic Quadrupedal Robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 38–44, October 2016.

[67] Jemin Hwangbo, Carmine Dario Bellicoso, Péter Fankhauser, and Marco Hutter. Probabilistic foot contact estimation by fusing information from dynamics and differential/forward kinematics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2016.

[68] KVH Industries. http://www.kvh.com/Commercial-and-OEM/Gyros-and-Inertial-Systems-and-Compasses/G yros-and-IMUs-and-INS/IMUs/1775-IMU.aspx.

[69] Michal Irani and P. Anandan. About direct methods. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, ICCV '99, pages 267–277, London, UK, UK, 2000. Springer-Verlag.

[70] Simon J. Julier and Jeffrey K. Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal Processing, Sensor Fusion, and Target Recognition VI*, volume 3068, pages 182–193, 1997.

[71] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J. Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

[72] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[73] R. Kṁmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3607–3613, May 2011.

[74] J.Z. Kolter, Youngjun Kim, and A.Y. Ng. Stereo vision and terrain modeling for quadruped robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1557–1564, 5 2009.

[75] T. Koolen, J. Smith, G. Thomas, S. Bertrand, J. Carff, N. Mertins, D. Stephen, P. Abeles, J. Englsberger, S. McCrory, J. van Egmond, M. Griffioen, M. Floyd, S. Kobus, N. Manor, S. Alsheikh, D. Duran, L. Bunch, E. Morphis, L. Colasanto, K. L. H. Hoang, B. Layton, P. Neuhaus, M. Johnson, and J. Pratt. Summary of team ihmc's virtual robotics challenge entry. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 307–314, October 2013.

[76] Twan Koolen, Sylvain Bertrand, Gray Thomas, Tomas de Boer, Tingfan Wu, Jesper Smith, Johannes Englsberger, and Jerry Pratt. Design of a Momentum-Based Control Framework and Application to the Humanoid Robot Atlas. *International Journal of Humanoid Robotics*, 13(1):1650007, 2016.

[77] J. H. Kotecha and P. M. Djuric. Gaussian particle filtering. *IEEE Transactions on Signal Processing*, 51(10):2592–2601, October 2003.

[78] Eric Krotkov, John Bares, Takeo Kanade, Tom Mitchell, Reid Simmons, and William (Red) L. Whittaker. Ambler: a six-legged planetary rover. In *Fifth International Conference on Advanced Robotics (ICAR)*, volume 1, pages 717 – 722, June 1991.

[79] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza. Low-latency visual odometry using event-based feature tracks. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 16–23, October 2016.

[80] Pei-Chun Lin, Haldun Komsuoglu, and Daniel E Koditschek. Sensor data fusion for body state estimation in a hexapod robot with dynamical gaits. *IEEE Transactions on Robotics*, 22(5):932–943, 2006.

[81] E. Lubbe, D. Withey, and K.R. Uren. State estimation for a hexapod robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6286–6291, September 2015.

[82] A. D. Luca, A. Albu-Schaffer, S. Haddadin, and G. Hirzinger. Collision detection and safe reaction with the dlr-iii lightweight manipulator arm. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1623–1630, October 2006.

[83] Jeremy Ma, Max Bajracharya, Sara Susca, Larry Matthies, and Matt Malchano. Real-time pose estimation of a dynamic quadruped in GPS-denied environments for 24-hour operation. *The International Journal of Robotics Research*, 2015.

[84] Jeremy Ma, Sara Susca, Max Bajracharya, Larry Matthies, Matt Malchano, and Dave Wooden. Robust Multi-Sensor, Day/Night 6-DOF Pose Estimation for a Dynamic Legged Vehicle in GPS-denied Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 619–626, May 2012.

[85] Martin Magnusson. *The Three-Dimensional Normal-Distributions Transform — an Efficient Representation for Registration, Surface Analysis, and Loop Detection.* PhD thesis, Örebro University, December 2009. Örebro Studies in Technology 36.

[86] Martin Magnusson, Henrik Andreasson, Andreas Nüchter, and Achim J. Lilienthal. Automatic appearance-based loop detection from 3D laser data using the normal distributions transform. *Journal of Field Robotics*, 26(11–12):892–914, November 2009.

[87] Mark Maimone, Yang Cheng, and Larry Matthies. Two years of Visual Odometry on the Mars Exploration Rovers. *Journal of Field Robotics*, 24(3):169–186, 2007.

[88] Mark W Maimone, P Chris Leger, and Jeffrey J Biesiadecki. Overview of the mars exploration rovers' autonomous mobility and vision capabilities. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2007.

[89] F. L. Markley. Attitude determination using vector observations and the singular value decomposition. *The Journal of the Astronautical Science*, 36(3):245–258, 1988.

[90] C. Mastalli, I. Havoutis, A. W. Winkler, D. G. Caldwell, and C. Semini. On-line and on-board planning and perception for quadrupedal locomotion. In *IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–7, May 2015.

[91] LORD Microstrain. http://www.microstrain.com/inertial/3dm-gx3-25.

[92] LORD Microstrain. http://www.microstrain.com/inertial/3dm-gx4-25.

[93] Hans Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.* PhD thesis, Stanford University, Sep 1980.

[94] Raúl Mur-Artal, JosḾ. M. Montiel, and Juan D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, October 2015.

[95] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *arXiv preprint arXiv:1610.06475*, 2016.

[96] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Proc. of the 2011 10th IEEE Int. Symp on Mixed and Augmented Reality*, ISMAR '11, pages 127–136, Washington, DC, USA, 2011. IEEE Computer Society.

[97] C.V. Nguyen, S. Izadi, and D. Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pages 524–530, October 2012.

[98] Matthias Nießner, Angela Dai, and Matthew Fisher. Combining inertial navigation and icp for real-time 3d surface reconstruction. *Eurographics (EG)*, 2014.

[99] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P.T. Furgale, and R. Siegwart. A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 431–437, 5 2014.

[100] David Nistér. An Efficient Solution to the Five-Point Relative Pose Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–777, June 2004.

[101] Simona Nobili, Raluca Scona, Marco Caravagna, and Maurice Fallon. Automatic ICP-Tuning for Robust Localization of a Humanoid Robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2017.

[102] E. Olson. A passive solution to the sensor synchronization problem. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1059–1064, October 2010.

[103] Romeo Orsolino, Michele Focchi, Darwin G. Caldwell, and Claudio Semini. An asymmetric model for quadrupedal bounding in place. In *9th International Workshop on Human-Friendly Robotics (HFR)*, 2016.

[104] E.A. Parr. Digital control systems. In M.A. Laughton and D.J. Warne, editors, *Electrical Engineer's Reference Book*, pages 1 – 36. Newnes, 16 edition, 2003.

[105] Pei-Chun Lin, H. Komsuoglu, and D.E. Koditschek. A leg configuration measurement system for full-body pose estimates in a hexapod robot. *IEEE Transactions on Robotics*, 21(3):411–422, June 2005.

[106] Pei-Chun Lin, H. Komsuoglu, and D.E. Koditschek. Sensor Data Fusion for Body State Estimation in a Hexapod Robot with Dynamical Gaits. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 22, pages 4733–4738, 2005.

[107] R. Playter, M. Buehler, and M. Raibert. BigDog. 6230:62302O, 2006.

[108] François Pomerleau, Francis Colas, and Roland Siegwart. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends® in Robotics*, 4(1):1–104, 2015.

[109] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing ICP variants on real-world data sets. *Autonomous Robots*, 34(3):133–148, February 2013.

[110] E. J. Post. Sagnac effect. *Reviews of Modern Physics*, 39:475–493, April 1967.

[111] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[112] Usman Qayyum and Jonghyuk Kim. Inertial-kinect fusion for outdoor 3d navigation. In *Australasian Conference on Robotics and Automation 2013 (ACRA 2014)*, 12 2013.

[113] B. S. Reddy and B. N. Chatterji. An fft-based technique for translation, rotation, and scale-invariant image registration. *IEEE Transactions on Image Processing*, 5(8):1266–1271, August 1996.

[114] M. Reinstein and M. Hoffmann. Dead reckoning in a dynamic quadruped robot: Inertial navigation system aided by a legged odometer. In *IEEE International Conference on Robotics and Automation*, pages 617–624, May 2011.

[115] Michal Reinstein and Matej Hoffmann. Dead reckoning in a dynamic quadruped robot based on multimodal proprioceptive sensory information. *IEEE Transactions on Robotics*, 29(2):563–571, 2013.

[116] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1515, October 2005.

[117] G. P. Roston and E. P. Krotkov. Dead reckoning navigation for walking robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 607–612, July 1992.

[118] Gerald P Roston and Eric P Krotkov. Dead Reckoning Navigation for Walking Robots. DTIC Document CMU-RI-TR-91-27, Carnegie Mellon University, November 1991.

[119] Nicholas Rotella, Michael Bloesch, Ludovic Righetti, and Stefan Schaal. State estimation for a humanoid robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 952–958. IEEE, 2014.

[120] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Third International Conference on 3-D Digital Imaging and Modeling (3DIM 2001)*, pages 145–152, 2001.

[121] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–4, May 2011.

[122] U. Saranli, M. Buehler, and D. E. Koditschek. Rhex: A simple and highly mobile hexapod robot. *International Journal of Robotics Research*, 20(7):616–631, 2001.

[123] Philippe Sardain and Guy Bessonnet. Forces acting on a biped robot. center of pressure-zero moment point. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 34(5):630–637, 2004.

[124] Paul G Savage. Strapdown Inertial Navigation Integration Algorithm Design Part 1: Attitude Algorithms. *Journal of Guidance, Control, and Dynamics*, 21(1):19–28, 1998.

[125] Paul G Savage. Strapdown Inertial Navigation Integration Algorithm Design Part 2: Position and Velocity Algorithms. *Journal of Guidance, Control, and Dynamics*, 21(2):19–28, January 1998.

[126] Abraham Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[127] D. Scaramuzza and F. Fraundorfer. Visual odometry [tutorial]. *IEEE Robotics Automation Magazine*, 18(4):80–92, December 2011.

[128] K. Schmid and H. Hirschmuller. Stereo vision and imu based real-time ego-motion and depth image computation on a handheld device. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4671–4678, May 2013.

[129] Aleksandr V Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-ICP. *Proceedings of Robotics: Science and Systems*, 2:4, 2009.

[130] Claudio Semini, Victor Barasuol, Jake Goldsmith, Marco Frigerio, Michele Focchi, Yifu Gao, and Darwin G. Caldwell. Design of the hydraulically-actuated, torque-controlled quadruped robot HyQ2Max. *IEEE/ASME Transactions on Mechatronics*, 2016.

[131] Claudio Semini, Nikos G Tsagarakis, Emanuele Guglielmino, Michele Focchi, Ferdinando Cannella, and Darwin G Caldwell. Design of hyq– a hydraulically and electrically actuated quadruped robot. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 2011.

[132] Claudio Semini, Nikos G. Tsagarakis, Emanuele Guglielmino, Michele Focchi, Ferdinando Cannella, and Darwin G. Caldwell. Design of HyQ - a hydraulically and electrically actuated quadruped robot. *J. of Systems and Control Engineering*, 2011.

[133] Jacopo Serafin and Giorgio Grisetti. NICP: Dense normal based point cloud registration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 742–749. IEEE, Sep 2015.

[134] B. Siciliano and O. Khatib. *Springer Handbook of Robotics*. Springer Handbook of Robotics. Springer Berlin Heidelberg, 2008.

[135] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*, chapter 2. MIT Press, 2 edition, 2011.

[136] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*, chapter 5. MIT Press, 2 edition, 2011.

[137] A. Stelzer, H. Hirschmuller, and M. Gorner. Stereo-vision-based navigation of a six-legged walking robot in unknown rough terrain. *The International Journal of Robotics Research*, 31(4):381–402, 2 2012.

[138] Benjamin J. Stephens. State estimation for force-controlled humanoid balance using simple models in the presence of modeling error. In *IEEE International Conference on Robotics and Automation*, pages 3994–3999. IEEE, May 2011.

[139] Alex Teichman, Stephen Miller, and Sebastian Thrun. Unsupervised intrinsic calibration of depth sensors via slam. In *Robotics: Science and Systems*, 2013.

[140] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*, chapter 3. MIT Press, 2005.

[141] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*, chapter 4. MIT Press, 2005.

[142] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical Report TR 95-041, University of North Carolina at Chapel Hill, 2006.

[143] Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John J. Leonard, and John McDonald. Real-time large scale dense rgb-d slam with volumetric fusion. *The International Journal of Robotics Research*, 34(4-5):598–626, April 2015.

[144] A. Winkler, I. Havoutis, S. Bazeille, J. Ortiz, M. Focchi, R. Dillmann, D. G. Caldwell, and C. Semini. Path planning with force-based foothold adaptation and virtual model control for torque controlled quadruped robots. In *IEEE ICRA*, 2014.

[145] A. W. Winkler, C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini. Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5148–5154, May 2015.

[146] Alexander Winkler, Ioannis Havoutis, Stephane Bazeille, Jesus Ortiz, Michele Focchi, Darwin G. Caldwell, and Claudio Semini. Path planning with force-based foothold adaptation and virtual model control for torque controlled quadruped robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[147] Oliver J. Woodman. An introduction to inertial navigation. Technical Report 696, University of Cambridge, August 2007.

[148] X. Xinjilefu, S. Feng, W. Huang, and C. G. Atkeson. Decoupled state estimation for humanoids using full-body dynamics. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 195–201, May 2014.

[149] X Xinjilefu, Siyuan Feng, and Christopher G. Atkeson. Dynamic State Estimation using Quadratic Programming. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 989–994, 2014.

[150] P. Zaytsev, S. J. Hasaneini, and A. Ruina. Two steps is enough: No need to plan far ahead for walking balance. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6295–6300, May 2015.