TUM Institute for Cognitive Systems (ICS)

OpenWalker

Framework Architecture

Emmanuel Dean, Florian Bergner, Rogelio Guadarrama-Olvera,
Simon Armleder, and Gordon Cheng

February 14, 2020

1 Project Description

1.1 Problem Statement

Biped and humanoid robots are complex systems composed by a large number of actuated degrees of freedom and various sensors that provide the requirements for the complex process of walking. Due to such complexity and number of components, in the past decades, the community of humanoid robots has developed several robot architectures for different purposes and applications. All these robot architectures, ranging from small-size hobby robots to giant manned walking machines, share some anatomical similarities developed through years of work in this field. Nevertheless, the task of coordinating the information generated by all the sensors and actuators in humanoid robots poses a complex software problem which has been approached with different paradigms. Consequently, a large variety of different software frameworks has been created which, in most of the cases, are intrinsically connected to the robot hardware that they were designed for. This hardware dependency makes it difficult to develop software packages which can be reused by other teams. As a result, the robot developers must continuously re-implement essential software modules/components to comply with the requirements and restrictions of their system, slowing down the progress of biped and humanoid robotics. In particular, two of the most essential components required by this type of robots are the balance and walking controllers.

1.2 Innovation

From the experiences of different researchers around the world [1], the need for a common middleware for handling the resources (sensors and actuators) of biped and humanoid robots is clear. In this context, ROS based software frameworks, such as ros_control [4], provide the

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

low-level hardware abstraction layer to read and write data to the sensors and actuators separating the control development level from the hardware interfaces while keeping all the benefits of the ROS middleware. However, for complex control architectures such as balance/walking controllers, there are no general frameworks to provide all the required components. Even though there are some packages available to build balance and walking controllers, such as foot-step planners, kinematic and dynamic libraries, state estimators, etc., they use different interfaces (even inside ROS) and it is not straightforward to implement walking controllers with them. Therefore, there is a need for an open source balance/walking controller which states the base for all the components required for balance and walking [3]. OpenWalker's aim is to satisfy the necessity of a framework for balance and walking controllers, and to easily adapt to any biped robot architecture. OpenWalker will provide the tools for developers and contributors to implement their own walking controllers without having to create the whole framework for every robot they use. OpenWalker will reduce the implementation time for new controllers and provide a clear implementation example to new researchers on the field of walking controllers. OpenWalker will also reduce the training time for new hobbyists and researchers and provide a common software structure to further develop different options for each component. Furthermore, OpenWalker can contribute to define a common framework to standarize controllers for humanoid robots, allowing to compare different control techniques.

1.3 Background

Biped walking is a challenging control problem. Its difficulty lies in several factors such as the inherently unstable dynamics of biped systems, the discrete time nature of footstep generation, the high number of degrees of freedom of biped robots, and the inability of directly measuring the floating base state. Consequently, walking controllers are hybrid discrete-continuous state machines which generate the motions for a stable gait while keeping the balance metrics under safe and feasible conditions [5]. Due to the complexity and the high number of degrees of freedom of biped and humanoid robots, a common strategy to generate walking motions is to consider simplified lower dimensional models for the robot. For example, one popular and simple model is the linear inverted pendulum. In such a model, the robot is considered as a concentrated mass attached to an inverted pendular beam which oscillates from one footstep position to the next one. Then, considering these simplified dynamics, a balance controller must provide the feedback law to bring the pendulum to an equilibrium point while a walking controller must push the pendulum away from its equilibrium point to transfer the mass to the equilibrium point of the next foot position. The coordinated push and pull motion produces the required cadence of a stable gait [6, 7]. Different approaches have been proposed to generate walking motion patterns using different simplified models. For example, the telescopic compass approach in [8] where the robot is modelled as a compass with telescopic legs or the divergent component of motion controller for the inverted pendulum model in [9], or the inverted pendulum with full centroidal inertia in [10]. Even when there is a large variety in approaches, most of the walking controllers based in simplified models share common components and similar architectures. Walking controllers such as [8, 9, 10] can be implemented in software architectures similar to the ones described in [11, 12]. They require a component to estimate the Zero Moment Point (ZMP) from sensor data [2], another component to estimate the position of the Centre of Mass (CoM), a component to compute the forward kinematics, a component to compute inverse kinematics, a footstep planner module, and a walking motion generation component (simplified model engine). These components can be implemented in different manners but the inputs and outputs are common metrics in

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

the biped locomotion theory. Some examples of ROS implementations of these components are: the ZMP-CoM components in the multi-contact-zmp packages [13], the footstep planner and localization modules in humanid_navigation stack [14], and the kinematic and dynamic libraries [15, 16]. There are also ROS-based robot-specific complete walking controllers published such as the controller from [11] for the iCub robot and [17] for the HRP-4 robot. However, these examples are either robot specific or based on advanced complex techniques and are not straightforward to adapt for other robots.

1.4 Objectives

OpenWalker will aim to fulfil the need of a standard walking controller architecture providing the minimal components required to generate stable walking on any biped robot with planar feet by adjusting a minimal set of parameters. Therefore, OpenWalker will provide the required components to handle the most common biped robot architectures for different joint interfaces (position velocity and effort) and to exploit the available sensors in common biped robot architectures (IMU sensors and ankle force-torque sensors). The main objective of this project is to provide a general open source control framework, which is easy to use, and well-documented to simplify the implementation of walking and balancing controllers. In this way, the main contribution of OpenWalker is the development of software tools, templates, and modules.

1.5 Expected Outcome

OpenWalker will provide a set of open source packages to enable walking on common biped robot architectures with minimum user input information such as the robot descriptor URDF files and a minimal set of parameters such as desired hip height, foot sole reference frames in the kinematic chain, and available sensors in the robot. All the components will be kept as general as possible to give users the freedom to modify them with low effort in order to test new balance and locomotion methods, reducing the implementation-test times. OpenWalker will enable users to test their walking controllers in an increasing number of simulated and real biped robots. The number of available robots will grow as other robot models are implemented under the ros_control hardware interface.

1.6 Technical Approach

Figure 1.1 shows the general pipe-line of OpenWalker. The entire project will be based on ros_control, and it will provide modules and templates for fast prototyping of balancing/walking controller components. The modules developed in this project will be used mainly for control, which requires real-time capabilities. Therefore, the modules and templates will be mainly written in C++. However, when possible, python code, examples, and templates will be also provided. The main components of OpenWalker are:

URDF Interface: This component extracts important information required for balancing and walking control algorithms from standard URDF files. The component also allocates this information in easy-to-use data containers (C++ classes), which provide simple access functions, allowing the transport of data in a self-contained form. Forward Kinematics component (FK): This component will use information generated by and contained in the URDF interface class. The main goal of this component is to abstract the complexity of standard Kinematic libraries,

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

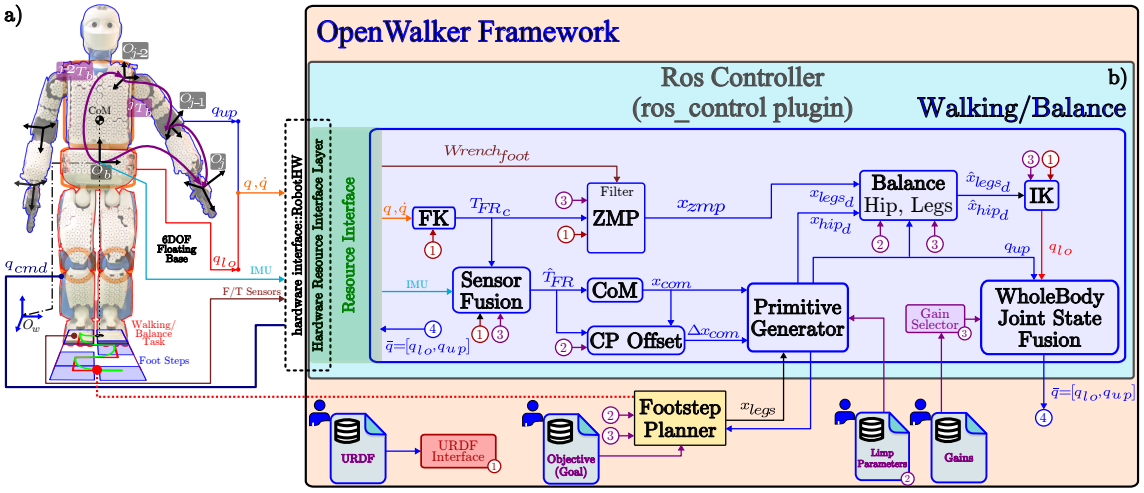

Figure 1.1: OpenWalker Control Framework

e.g. KDL, and allow the user to focus on the required information and not on how this information needs to be extracted. The information will be provided through homogenized interfaces and data containers defined within the OpenWalker software architecture. Sensor Fusion component: This component will deal with the problem of the floating-base state estimation. It will provide tools to implement algorithms to fuse information from different sensors to compute the floating-base state in real-time. This component will automatically extract the information provided by the URDF files and user-defined gains to simplify this process.

Zero Moment Point (ZMP) component: This component will estimate the Zero Moment Point of the supporting legs from Force-Torque sensor data. It will be also customizable to receive feedback from other kinds of plantar sensors as in [18]. Center of Mass (CM) component: This component will compute the CM of the robot using the current joint state and the mechanical information provided in the URDF files of the robot. Capture Point (CP) component: This component provides methods to compute the CP . The main algorithm to compute the CP is described in [6]. This component will provide a user-interface to change the parameters in a simple form.

Footstep Planner component: This module is an external module, which means it will not be implemented as a ros_control plugin. However, it will be implemented as a standard ros node. This module will use the limb parameters, gains, and the target goal for the robot (all this information is defined by the user) to generate a discrete sequence of footsteps. This sequence will be the reference (desired) foot poses needed for the controllers. The main concept of this module is to abstract the complexity of the footstep design and provide standard and ready to use footstep planners. This module is mainly focused on the lower limbs of the humanoid robot.

Primitive Generator component: This is one of the most important components of this framework since it generates adequate footstep motions (in continuous space) that take into account the desired discrete foot poses and the current state of the robot. Even when this module is

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

mainly focused on the lower-limbs, it will also provide standard upper-limbs sequences used commonly in the literature, e.g. standard swinging arms based on the gait frequency.

Balance component: In the same form as the last module, this component is paramount for the whole framework. This module uses the reference trajectories of the hip and legs computed by the primitive generation component and includes the forces acting on the robot. In this manner, this component will handle the problem of providing dynamically consistent joint hip and legs reaction motions to guarantee a stable walking pattern rejecting disturbances from external and internal forces. This component will also use the information provided by the user to simplify the parameterization of the balancer.

Inverse Kinematic (IK) component: This component will provide standard IK solvers with especial emphasis on the methods which are more suitable for bipedal walking. The main goal is to provide ready-to-use IK applications which are robust and real-time ready. This component will abstract the complexity of the implementation of state-of-the-art IK solvers included in standard IK libraries.

Whole-Body Joint State component: This component will receive the upper and lower joint trajectories and will include the system constraints to generate adequate joint commanded trajectories. This is an important step since the joint positions commanded to the robot should also take into account the hardware limitations to guarantee that the joint motions are realizable.

Resource Interface component: Where the ros_control framework provides a low-level abstraction layer for the hardware, this component provides a high-level abstraction. This interface component handles the information exchange between ros_control and the OpenWalker modules using the data containers defined of our framework. In this manner, the information exchange between component is homogenized and synchronized. Furthermore, the resource interface simplifies the maintenance of the OpenWalker component since changes in ros_control only affect the resource interface module, and end-users who want to use the OpenWalker without ros_control can optionally develop custom interfaces for more complex behaviours as [19].

The OpenWalker framework will be fully integrated in the ROS ecosystem and it will be validated in different humanoid robot platforms, e.g. REEM-C (PAL), TALOS (PAL), BIOLOID (Robotis), etc. During these tests, a comprehensive documentation will be generated to allow both experts and non-experts to develop their own examples and modules. The documentation will include an extensive list of examples that range from simple tutorials to more advanced applications. The OpenWalker framework, documentation, and tutorials will be released in the ROS repositories, making them freely available to the whole community.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

2 Framework Architecture

For the design the Framework architecture we extensively refined our general overview of the control framework introduced in Figure 1.1 towards a system of inter-connected modules. These modules encapsulate implementable/programmable functional blocks and we describe in the following the behavior, the employed methods, the inputs and outputs, and the dependencies of each module.

Furthermore, we defined the nomenclature and conventions we employ during the development of the framework to ensure consistency and homogeneous code generation throughout the whole project. The design of the core components of the framework which include type classes to enforce strong type coding, and abstract interface classes to separate the interfaces (inputs/outputs/actions) of the modules from their implementation.

The overview of the framework with all modules and their connections can be found in Figure 2. Table 2 summarizes the most important type classes and Table 2.2 provides an overview of the base classes derived from types of the Eigen math library and the default parameterization of the employed template classes.

This document is followed by the description of the naming conventions and the module descriptions.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

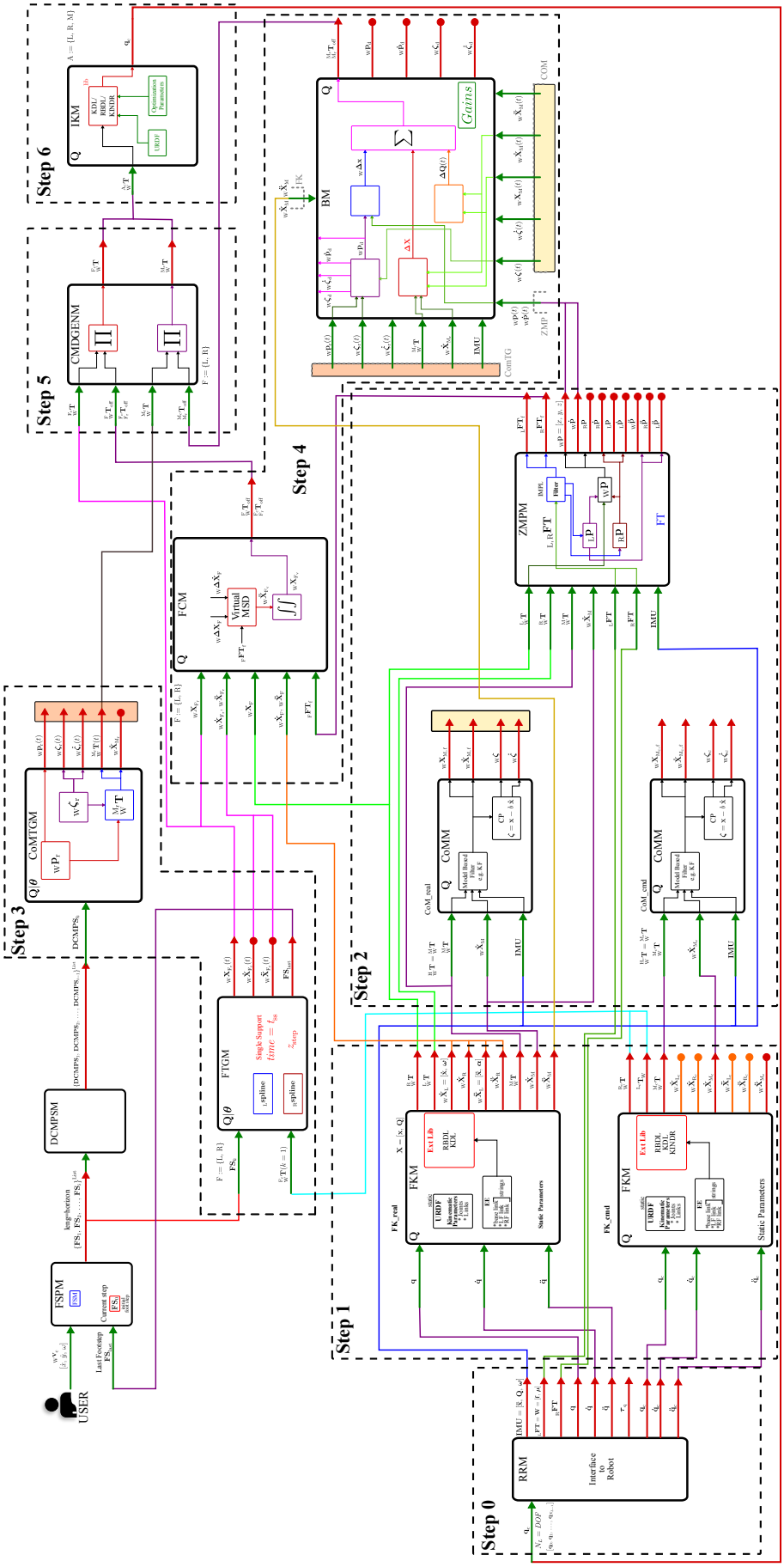

Figure 2.1: Overall overview of the OpenWalker Framework Architecture.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

Symbol	Name	Class Name	Typedef	Base Class
$\mathbf{q} \in \mathbb{R}^{DOF}$	Joint Position	JointPosition	JPos	VectorDof
$\dot{\mathbf{q}} \in \mathbb{R}^{DOF}$	Joint Velocity	JointVelocity	JVel	VectorDof
$\ddot{\mathbf{q}} \in \mathbb{R}^{DOF}$	Joint Acceleration	JointAcceleration	JAcc	VectorDof
$\boldsymbol{\tau}_{\mathbf{q}} \in \mathbb{R}^{DOF}$	Joint Effort	JointEffort	JTorque	VectorDof
$\mathbf{x} \in \mathbb{R}^3$	Linear Position	LinearPosition	LPos	Vector3
$\dot{\mathbf{x}} \in \mathbb{R}^3$	Linear Velocity	LinearVelocity	LVel	Vector3
$\ddot{\mathbf{x}} \in \mathbb{R}^3$	Linear Acceleration	LinearAcceleration	LAcc	Vector3
$\mathbf{Q} \in \mathbb{H}$	Angular Position	AngularPosition	APos	Quaternion
$\boldsymbol{\omega} \in \mathbb{R}^3$	Angular Velocity	AngularVelocity	AVel	Vector3
$\boldsymbol{\alpha} = \dot{\boldsymbol{\omega}} \in \mathbb{R}^3$	Angular Acceleration	AngularAcceleration	AAcc	Vector3
$\mathbf{X} = [\mathbf{x}, \mathbf{Q}] \in \mathbb{R}^7$	Cartesian Position	CartesianPosition	CPos	Vector7
$\dot{\mathbf{X}} = [\dot{\mathbf{x}}, \boldsymbol{\omega}] \in \mathbb{R}^6$	Cartesian Velocity	CartesianVelocity	CVel	Vector6
$\ddot{\mathbf{X}} = [\ddot{\mathbf{x}}, \boldsymbol{\alpha}] \in \mathbb{R}^6$	Cartesian Acceleration	CartesianAcceleration	CAcc	Vector6
$\mathbf{T} \in \mathbb{R}^{4\times4}$	Homogeneous Transformation	HomogeneousTransformation	HTf	Transform3
$\mathbf{f} \in \mathbb{R}^3$	Force	Force	Force	Vector3
$\boldsymbol{\mu} \in \mathbb{R}^3$	Moment	Moment	Moment	Vector3
$\mathbf{W} = [\mathbf{f}, \boldsymbol{\mu}] \in \mathbb{R}^6$	Wrench	Wrench	Wrench	Vector6
$\mathbf{FT} = \mathbf{W} = [\mathbf{f}, \boldsymbol{\mu}] \in \mathbb{R}^6$	Force Torque Sensor	ForceTorqueSensor	FTSensor	Wrench
$\mathbf{IMU} = [\ddot{\mathbf{x}}, \mathbf{Q}, \boldsymbol{\omega}]$	Inertial Measurement Unit Sensor	ImuSensor	ImuSensor	[LAcc, APos, AVel]
$\mathbf{p} \in \mathbb{R}^3$	Zero Moment Point (ZMP)	ZeroMomentPoint	Zmp	LinearPosition
$\dot{\mathbf{p}} \in \mathbb{R}^3$	ZMP Velocity	ZeroMomentPointP	ZmpP	LinearVelocity
$\ddot{\mathbf{p}} \in \mathbb{R}^3$	ZMP Acceleration	ZeroMomentPointPP	ZmpPP	LinearAcceleration
$\boldsymbol{\zeta} \in \mathbb{R}^3$	Divergent Component of Motion (DCM)	DivergentComponentOfMotion	Dcm	LinearPosition
$\dot{\boldsymbol{\zeta}} \in \mathbb{R}^3$	DCM Velocity	DivergentComponentOfMotionP	DcmPP	LinearVelocity
$\ddot{\boldsymbol{\zeta}} \in \mathbb{R}^3$	DCM Acceleration	DivergentComponentOfMotionPP	DcmPP	LinearAcceleration
$\mathbf{r}_i \in \mathbb{R}^3$	Virtual Repellent Point (VRP)	VirtualRepellentPoint	Vrp	LinearPosition
$\boldsymbol{\zeta}_i \in \mathbb{R}^3$	DCM way point (DCMWP)	DcmWayPoint	DcmWp	LinearPosition
$\boldsymbol{\zeta}_{\text{DS,ini},i} \in \mathbb{R}^3$	Initial double support way point for the DCM	DcmDoubleSupportInitialWayPoint	DcmDsIWp	LinearPosition
$\boldsymbol{\zeta}_{\text{DS,final},i} \in \mathbb{R}^3$	Final double support way point for the DCM	DcmDoubleSupportFinalWayPoint	DcmDsFWp	LinearPosition
$\dot{\boldsymbol{\zeta}}_{\text{DS,ini},i} \in \mathbb{R}^3$	Velocity of the initial double support way point for the DCM	DcmDoubleSupportInitialWayPointP	DcmDsIWpP	LinearVelocity
$\dot{\boldsymbol{\zeta}}_{\text{DS,final},i} \in \mathbb{R}^3$	Velocity of the final double support way point for the DCM	DcmDoubleSupportFinalWayPointP	DcmDsFWpP	LinearVelocity
$\mathbf{DCMPS} = \left[\mathbf{r}_i, \boldsymbol{\zeta}_i, \boldsymbol{\zeta}_{\text{DS,ini},i}, \boldsymbol{\zeta}_{\text{DS,final},i}, \dot{\boldsymbol{\zeta}}_{\text{DS,ini},i}, \dot{\boldsymbol{\zeta}}_{\text{DS,final},i}\right]$	DCM Point Set	DcmPointSet	DcmPs	[Vrp, DcmWp, DcmDsIWp, DcmDsFWp, DcmDsIWpP, DcmDsFWpP]
$\mathbf{X}_{\text{FS},i} \in \mathbb{R}^7$	Foot Step Cartesian Position	FootStepCartesianPosition	FsCPos	CartesianPosition
$l \in \{\text{L}, \text{R}\}$	Leg Identifier (Left/Right leg)	FootStepLegIdentifier	FsLId	LegIdentifier
$s_i \in \{0, 1\}$	Foot Step End Flag	FootStepEndFlag	FsEndF	bool
$n_i \in \mathbb{N}_0^+$	Foot Step Number	FootStepNumber	FsNum	unsigned int
$\mathbf{FS} = \left[\mathbf{X}_{\text{FS},i}, l_i, s_i, n_i\right]$	Foot Step	FootStep	Fs	[FSCPos, FsLId, FsEndF, FsNum]

Table 2.1: Overview of type classes used in the OpenWalker project.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

Base Class	Eigen Template Class	Default Template Parameters
VectorDof	Eigen::Matrix<Scalar, Dof, 1>	Scalar = double Dof = Eigen::Dynamic
Vector3	Eigen::Matrix<Scalar, 3, 1>	Scalar = double
Vector6	Eigen::Matrix<Scalar, 6, 1>	Scalar = double
Vector7	Eigen::Matrix<Scalar, 7, 1>	Scalar = double
Transform3	Eigen::Transform<Scalar, 3, Eigen::Affine>	Scalar = double
Quaternion	Eigen::QuaternionBase<Eigen::QuaternionRef<Derived> >	Derived = Eigen::Matrix<Scalar,4,1> Scalar = double

Table 2.2: Overview of base classes derived from types of the Eigen math library and the default parameterization of the employed template classes

References

[1] Natale, L., Asfour, T., Kanehiro, F., and Vahrenkamp, N. (Eds.). (2018). Software Architectures for Humanoid Robotics. Frontiers Media SA.

[2] Kajita, S., Hirukawa, H., Harada, K., and Yokoi, K. (2014). Introduction to humanoid robotics (Vol. 101). Springer Berlin Heidelberg.

[3] Grizzle, J. W., Chevallereau, C., Sinnet, R. W., and Ames, A. D. (2014). Models, feedback control, and open problems of 3D bipedal robotic walking. Automatica, 50(8), 1955-1988.

[4] Chitta, S., Marder-Eppstein, E., Meeussen, W., Pradeep, V., Tsouroukdissian, A. R., Bohren, J., ... and Perdomo, E. F. (2017). ros_control: A generic and simple control framework for ROS. The Journal of Open Source Software, 2(20), 456-456.

[5] Miyakoshi, S., and Cheng, G. (2003). Utilizing Physical Relationships for Biped Walking Control: a preliminary study in identifying key essential properties for the two support phases. In Proc. of Int. Conf. on Climbing and Walking Robots (pp. 543-550).

[6] Morimoto, J., Endo, G., Nakanishi, J., Hyon, S., Cheng, G., Bentivegna, D., and Atkeson, C. G. (2006, May). Modulation of simple sinusoidal patterns by a coupled oscillator model for biped walking. In Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006. (pp. 1579-1584). IEEE.

[7] Endo, G., Nakanishi, J., Morimoto, J., and Cheng, G. (2005, April). Experimental studies of a neural oscillator for biped locomotion with QRIO. In Proceedings of the 2005 IEEE international conference on robotics and automation (pp. 596-602). IEEE.

[8] Miyakoshi, S., and Cheng, G. (2004, October). Examining human walking characteristics with a telescopic compass-like biped walker model. In 2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583) (Vol. 2, pp. 1538-1543). IEEE.

[9] Englsberger, J., Ott, C., Roa, M. A., Albu-Schäffer, A., and Hirzinger, G. (2011, September). Bipedal walking control based on capture point dynamics. In 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 4420-4427). IEEE.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

[10] Lee, S. H., and Goswami, A. (2007, April). Reaction mass pendulum (RMP): An explicit model for centroidal angular momentum of humanoid robots. In Proceedings 2007 IEEE International Conference on Robotics and Automation (pp. 4667-4672). IEEE.

[11] Choi, Y., Kim, D., Oh, Y., and You, B. J. (2007). Posture/walking control for humanoid robot based on kinematic resolution of com jacobian with embedded motion. IEEE Transactions on Robotics, 23(6), 1285-1293.

[12] Park, I. W., Kim, J. Y., and Oh, J. H. (2006, December). Online biped walking pattern generation for humanoid robot khr-3 (kaist humanoid robot-3: Hubo). In 2006 6th IEEE-RAS International Conference on Humanoid Robots (pp. 398-403). IEEE.

[13] Caron, S., Pham, Q. C., and Nakamura, Y. (2017). Zmp support areas for multicontact mobility under frictional constraints. IEEE Transactions on Robotics, 33(1), 67-80.

[14] Hornung, A., Dornbush, A., Likhachev, M., and Bennewitz, M. (2012, November). Anytime search-based footstep planning with suboptimality bounds. In 2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012) (pp. 674-679). IEEE.

[15] Felis, M. L. (2017). RBDL: an efficient rigid-body dynamics library using recursive algorithms. Autonomous Robots, 41(2), 495-511.

[16] Smits, R., Bruyninckx, H., and Aertbeliën, E. (2011). Kdl: Kinematics and dynamics library.

[17] Caron, S., Kheddar, A., and Tempier, O. (2018). Stair climbing stabilization of the HRP-4 humanoid robot using whole-body admittance control. arXiv preprint arXiv:1809.07073.

[18] Guadarrama-Olvera, J. R., Bergner, F., Dean, E., and Cheng, G. (2018, November). Enhancing Biped Locomotion on Unknown Terrain Using Tactile Feedback. In 2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids) (pp. 1-9). IEEE.

[19] Dean-Leon, E., Guadarrama-Olvera, J. R., Bergner, F., Dean, E., and Cheng, G. (2019, Feb). Whole-Body Active Compliance Control for Humanoid Robots with Robot Skin. Accepted for 2019 IEEE International Conference on Robotics and Automation (ICRA). IEEE.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Naming Conventions

Emmanuel Dean, Florian Bergner, Rogelio Guadarrama-Olvera,
Simon Armleder, and Gordon Cheng

February 14, 2020

1 General description.

1.1 General coding rules

1.1.1 Formating

1. Blocks are intended by **2 spaces**
2. Braces are opened and closed in their own lines

1.1.2 Naming

1. ROS packages are **under_scored** e.g. `"my_package"`
2. ROS topics/Services are **under_scored** e.g. `"my_topic"`
3. Files are **under_scored** e.g. `"my_class.cpp"`
4. Libraries are **under_scored** e.g. `"my_class.cpp"`
5. Classes are **CamelCase** e.g. `"MyClass"`
6. Functions are **camelCased** e.g. `"myFunction()"`
7. Variables are **under_scored** e.g. `"my_var"`
8. Member Variables are **under_scored** with a trailing underscore e.g. `"my_member_var_"`
9. Global Variables are **under_scored** with a leading **g_** added e.g. `"g_my_global_var_"`
10. Namespaces are **under_scored** e.g. `"my_namespace"`

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

1.1.3 OW specific Naming

1. Code readability has higher priority than briefness.
2. Variable names for math expressions must follow the naming convention in Table 1.1, where "Op" is the acronym for "Operator" and "Qual" for "Qualifier".
3. The proposed notation is case sensitive.
4. Operator order must follow table 1.2.
5. Qualifier notation must follow table 1.3.

Table 1.1: General naming convension for math notation.

Name	Math	Code
Scalar	$name^{Op}_{\text{Qual, }index}$	`name<Op><Qual>< _ ><[index]>`
Vector	$\text{Base}\mathbf{name}^{\mathbf{Op1}\ Op2}_{\text{Qual, }index}$	`name<Op1><Op2><Qual>< _frame>< _base>< _ ><[index]>`
Matrix	$\mathbf{Name}^{\mathbf{Op1}\ Op2}_{\text{Qual, }index}$	`Name<Op1><Op2><Qual>< _ ><[index]>`
Homogeneous Transformation	$^{\text{Frame}}_{\text{Base}}\mathbf{Tname}^{Op}_{\text{Qual, }index}$	`Tname<Op><Qual>< _frame>< _base>< _ ><[index]>`

Table 1.2: Coding notation and order for operators.

Priority	Name	Math	Code
1	Derivative	$\dot{\bullet}$	`P`
2	Inverse	$\bullet^{-1}$ or $\bullet^{\dagger}$	`I`
3	Transposed	$\bullet^{\top}$	`T`

2 Variable glosary.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

Table 1.3: Coding notation for qualifiers.

Name	**Abbreviation**	**Code**	**Description**
Reference	ref	`Ref`	Reference value from abstract simplified model.
Desired	d	`D`	Desired quantity to be tracked by controller.
Real	real	`Real`	Real quantity from sensor data.
Commanded	cmd	`Cmd`	Value commanded to the actuators.

Math	**Code**	**Name**
q	`q`	Joint state
$\mathbf{q}_{\text{real}}$	`qReal`	Real joint state
$\mathbf{q}_{\text{cmd}}$	`qCmd`	Commanded joint state
${}_{\text{W}}\mathbf{x}_{\text{F}_\text{d}}$	`XD_f_w`	Desired position of **f** wrt **w**
${}_{\text{W}}\dot{\mathbf{x}}_{\text{F}_\text{d}}$	`XDP_f_w`	Desired velocity of **f** wrt **w**
${}_{\text{W}}\ddot{\mathbf{x}}_{\text{F}_\text{d}}$	`XDPP_f_w`	Desired acceleration of **f** wrt **w**

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Module Description: Foot Step Planner (FSPM)

Rogelio Guadarrama-Olvera, Emmanuel Dean, Florian Bergner,
Simon Armleder, and Gordon Cheng

February 14, 2020

1 Module Description

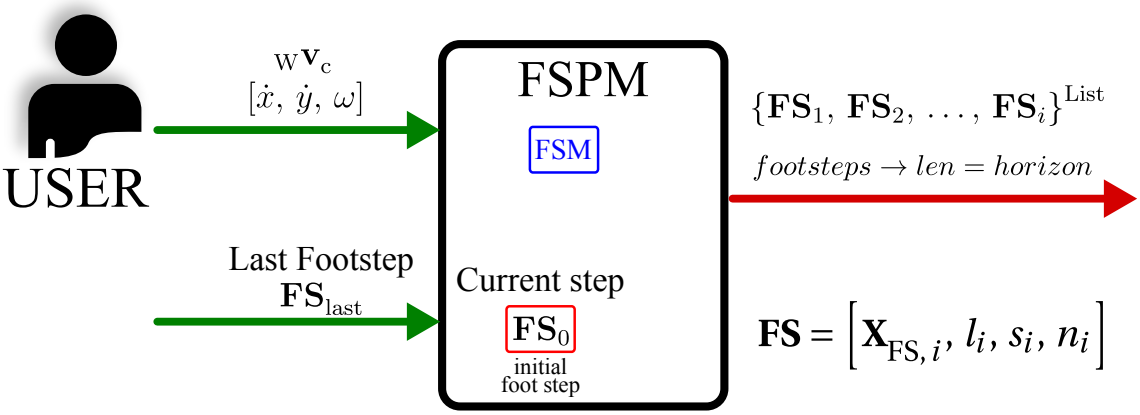

Figure 1.1: Foot Step Planner module: This module generates a set of footsteps from user velocity commands and adapts them according to the real footstep execution data.

The footstep planner generates a number of steps ahead proposing feasible footholds defined by the robot's kinematic parameters. The plan is generated from velocity commands defined by the user. The footstep planner also updates the plan when the step execution differs from the original plan. This could happen when the terrain conditions move the foot landing motion to a different location. Then the modified location is used to re-plan the following steps.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

2 Module Connections

2.1 Inputs

Symbol	Name	Type	Description
$\mathbf{v}_c \in \mathbb{R}^6$	Velocity Command	`CartesianVelocity`	Velocity command for the $\mathbf{x}-\mathbf{y}$ plane including rotation ω. The OpenWalker framework uses this module input to compute a feasible set of future footholds which enforce the walking execution to follow the velocity command.
FS	Last Footstep	`FootStep`	The end position of the last executed footstep. The OpenWalker framework uses this module input to adjust the plan of footsteps according to the actual execution of them.

2.2 Outputs

Symbol	Name	Type	Description
$\{\mathbf{FS}_i\}$	Footstep List	`FootStepList`	A list containing the planned footsteps.

2.3 Inter-Connections

This module can receive the input from any publisher of `geometry_msgs/Twist` messages, for example the `teleop_tools` packages. The Last Footstep is received from the Foot Trajectory Generator (FTG).

The output will provide the reference way-points for the DCM planner and the Foot Trajectory Generator (FTG).

2.4 Common Methods

There are several algorithmic methods for footstep planning using different sensors and map information. Some examples are the obstacle avoiding planner used in the humanoid robot ASIMO [1], the dynamic programming approach in [2], or the LIDAR based ROS packages `footstep_planner` and `visir_footstep_planner`.

References

[1] Chestnutt, Joel, et al. "Footstep planning for the honda asimo humanoid." Proceedings of the 2005 IEEE international conference on robotics and automation. IEEE, 2005.

[2] Kuffner, James J., et al. "Footstep planning among obstacles for biped robots." Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180). Vol. 1. IEEE, 2001.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Module Description: Real Robot (RRM)

Florian Bergner, Emmanuel Dean, Rogelio Guadarrama-Olvera,
Simon Armleder, and Gordon Cheng

February 14, 2020

1 Module Description

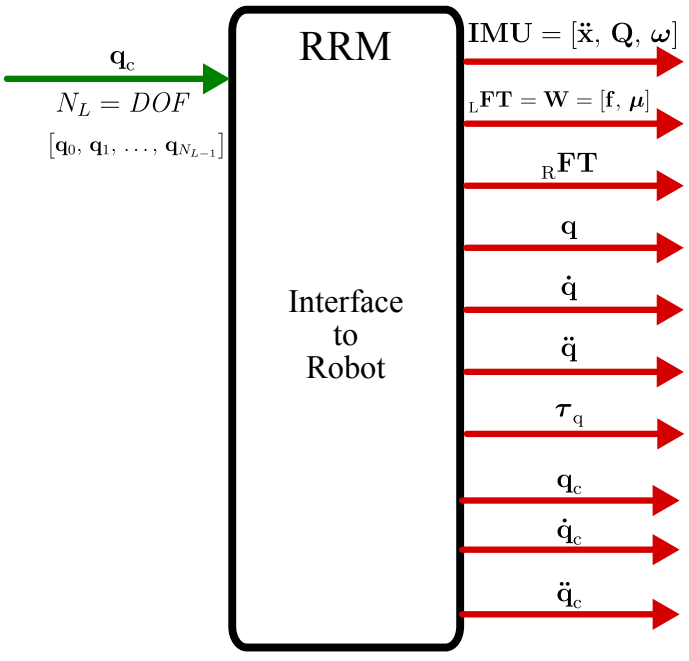

Figure 1.1: Real Robot module: This module implements the interface to the real robot.

The *Real Robot* module (RRM) implements the interface to the real robot. This module acquires all the available information provided by the sensors of the real robot and sends commands to the actuators of the robot. The RRM abstracts robot specific procedures for reading its

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

sensors and write commands to its actuators. Commands and sensor information are provided in the data types of the OpenWalker project and interfaces structure and control the access to the information. The module manages the access to the robot's sensors and actuators and homogenizes the exchanged information. As a result, all modules of the OpenWalker project can access different robots in the same way using the same data types. The common access eases development and maintenance.

2 Module Connections

2.1 Inputs

Symbol	Name	Type	Description
$\mathbf{q}_c \in \mathbb{R}^{DOF}$	Commanded Joint Position	`JointPosition`	This vector contains the next commanded joint positions for all the joints of the robot. The OpenWalker framework uses this module input to send position commands to the position controlled real robot.

2.2 Outputs

Symbol	Name	Type	Description
$\ddot{\mathbf{x}}_{imu} \in \mathbb{R}^3$	IMU Linear Acceleration	`LinearAcceleration`	This vector contains the linear acceleration measured by the Inertial Measurement Unit (IMU) sensor of the robot. The OpenWalker framework can use this module output for model based state estimations.
$\mathbf{Q}_{imu} \in \mathbb{R}^4$	IMU Angular Position	`AngularPosition`	This vector contains the angular position in quaternion measured by the Inertial Measurement Unit (IMU) sensor of the robot. The OpenWalker framework can use this module output for model based state estimations.
$\boldsymbol{\omega}_{imu} \in \mathbb{R}^3$	IMU Angular Velocity	`AngularVelocity`	This vector contains the angular velocity in quaternion measured by the Inertial Measurement Unit (IMU) sensor of the robot. The OpenWalker framework can use this module output for model based state estimations.
${}_L\mathbf{FT} = {}_L\mathbf{W} \in \mathbb{R}^6$	FT Left Foot Wrench	`ForceTorqueSensor`	This vector contains the wrench measured by the force torque sensor in the left foot of the robot. The OpenWalker framework can use this module output for the Zero-Moment-Point estimation.
${}_R\mathbf{FT} = {}_R\mathbf{W} \in \mathbb{R}^6$	FT Right Foot Wrench	`ForceTorqueSensor`	This vector contains the wrench measured by the force torque sensor in the right foot of the robot. The OpenWalker framework can use this module output for the Zero-Moment-Point estimation.
$\mathbf{q} \in \mathbb{R}^{DOF}$	Real Robot Joint Positions	`JointPosition`	This vector contains the real joint positions of the robot. The OpenWalker framework uses this module output to compute the forward kinematics of the real robot.
$\dot{\mathbf{q}} \in \mathbb{R}^{DOF}$	Real Robot Joint Velocities	`JointVelocity`	This vector contains the real joint velocities of the robot. The OpenWalker framework uses this module output to compute the forward kinematics of the real robot.
$\ddot{\mathbf{q}} \in \mathbb{R}^{DOF}$	Real Robot Joint Accelerations	`JointAcceleration`	This vector contains the real joint accelerations of the robot. The OpenWalker framework uses this module output to compute the forward kinematics of the real robot.
$\boldsymbol{\tau}_q \in \mathbb{R}^{DOF}$	Real Robot Joint Torques	`JointEffort`	This vector contains the real joint torques of the robot. The OpenWalker framework can use this module output to monitor the joint torques of the real robot.
$\mathbf{q}_c \in \mathbb{R}^{DOF}$	Commanded Robot Joint Positions	`JointPosition`	This vector contains the currently commanded joint positions of the robot. The OpenWalker framework uses this module output to compute the forward kinematics of the commanded robot.
$\dot{\mathbf{q}}_c \in \mathbb{R}^{DOF}$	Commanded Robot Joint Velocities	`JointVelocity`	This vector contains the currently commanded joint velocities of the robot. The OpenWalker framework uses this module output to compute the forward kinematics of the commanded robot.
$\ddot{\mathbf{q}}_c \in \mathbb{R}^{DOF}$	Commanded Robot Joint Accelerations	`JointAcceleration`	This vector contains the currently commanded joint accelerations of the robot. These are numerically computed by deriving $\mathbf{q}_c$. The OpenWalker framework uses this module output to compute the forward kinematics of the commanded robot.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

2.3 Inter-Connections

The RRM is connected to the two forward kinematics modules (FKMs) and provides the real and commanded joint positions, velocities, and acceleration to these modules. The real and commanded FKM need this information to compute/update the forward kinematics. The RRM also provides the IMU measurements to the Center-of-Mass module (CoMM), the Zero-Moment-Point module (ZMPM), and the Balancer module (BM) to provide additional information for state estimators. Additionally the RRM provides the FT sensor measurements to the ZMPM which filters the measurements. These filtered FT sensor measurements are then used by the ZMPM itself and provided to other modules.

2.4 Common Methods

This module is a pure interface module and thus does not need any mathematical, physical, or robotical models to compute its outputs. The main tasks of the RRM module are access management, type conversions, and structuring of information.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Module Description: Forward Kinematics (FKM)

Florian Bergner, Emmanuel Dean, Rogelio Guadarrama-Olvera,
Simon Armleder, and Gordon Cheng

February 14, 2020

1 Module Description

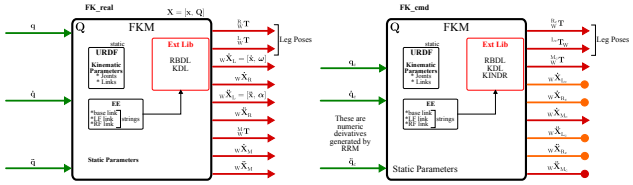

Figure 1.1: Forward Kinematics module: This module implements the forward kinematics for the robot.

The *Forward Kinematics* module (FKM) computes the forward kinematics of the robot. The OpenWalker framework employs two FKMs, one for computing the forward kinematics for the real robot, and one for the commanded robot. The forward kinematics maps the joint position space into the Cartesian space, i.e. for a given set of joint positions, the forward kinematics computes the Carthesian position (linear position and angular position) of a given end-effector. The forward kinematics of the real robot uses the robot's joint sensor measurements ($\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$) as input to compute the Carthesian positions of end-effectors of the real robot. Correspondingly, the forward kinematics of the commanded robot uses the commanded joint positions ($\mathbf{q}_c$, $\dot{\mathbf{q}}_c$, and $\ddot{\mathbf{q}}_c$) to compute the Carthesian positions of end-effectors of the commanded robot. The OpenWalker framework uses the real and commanded Carthesian end-effector positions to compute offsets which the frameworks uses to compensate the error between where it commanded the end-effectors and where they actually are.

Since the rigid multi body system (MBS) of the robot is the same for the real and the commanded robot the OpenWalker framework needs to realize only one FKM, which is then im-

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

plemented and connected once to the real joint positions and once to the commanded joint positions.

The OpenWalker framework requires the Carthesian position, velocities, and accelerations of three end-effectors, namely the left and right foot, and the center-of-mass (CoM) of the whole robot, with respect to the world. The FKM implementations provide this information to other modules of the OpenWalker project.

2 Module Connections

2.1 Inputs

Symbol	Name	Type	Description
$\mathbf{q} \in \mathbb{R}^{DOF}$	Real Robot Joint Positions	`JointPosition`	This vector contains the real joint positions of the robot. The OpenWalker framework uses this module input to compute the forward kinematics of the real robot.
$\dot{\mathbf{q}} \in \mathbb{R}^{DOF}$	Real Robot Joint Velocities	`JointVelocity`	This vector contains the real joint velocities of the robot. The OpenWalker framework uses this module input to compute the forward kinematics of the real robot.
$\ddot{\mathbf{q}} \in \mathbb{R}^{DOF}$	Real Robot Joint Accelerations	`JointAcceleration`	This vector contains the real joint accelerations of the robot. The OpenWalker framework uses this module input to compute the forward kinematics of the real robot.
$\mathbf{q}_c \in \mathbb{R}^{DOF}$	Commanded Robot Joint Positions	`JointPosition`	This vector contains the currently commanded joint positions of the robot. The OpenWalker framework uses this module input to compute the forward kinematics of the commanded robot.
$\dot{\mathbf{q}}_c \in \mathbb{R}^{DOF}$	Commanded Robot Joint Velocities	`JointVelocity`	This vector contains the currently commanded joint velocities of the robot. The OpenWalker framework uses this module input to compute the forward kinematics of the commanded robot.
$\ddot{\mathbf{q}}_c \in \mathbb{R}^{DOF}$	Commanded Robot Joint Accelerations	`JointAcceleration`	This vector contains the currently commanded joint accelerations of the robot. The OpenWalker framework uses this module input to compute the forward kinematics of the commanded robot.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

2.2 Outputs

FK of the Real Robot

Symbol	Name	Type	Description
${}^{L}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Left Foot Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix transforms coordinates in the left foot coordinate frame L to the world coordinate frame W.
${}^{R}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Right Foot Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix transforms coordinates in the right foot coordinate frame R to the world coordinate frame W.
${}^{M}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	CoM Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix transforms coordinates in the CoM coordinate frame M to the world coordinate frame W.
${}_{W}\dot{\mathbf{X}}_{L} \in \mathbb{R}^{6}$	Left Foot Velocity	`CartesianVelocity`	This vector contains the linear and angular velocities of the left foot L with respect to the world coordinate frame W.
${}_{W}\dot{\mathbf{X}}_{R} \in \mathbb{R}^{6}$	Right Foot Velocity	`CartesianVelocity`	This vector contains the linear and angular velocities of the right foot R with respect to the world coordinate frame W.
${}_{W}\dot{\mathbf{X}}_{M} \in \mathbb{R}^{6}$	CoM Velocity	`CartesianVelocity`	This vector contains the linear and angular velocities of the CoM with respect to the world coordinate frame W.
${}_{W}\ddot{\mathbf{X}}_{L} \in \mathbb{R}^{6}$	Left Foot Acceleration	`CartesianAcceleration`	This vector contains the linear and angular accelerations of the left foot L with respect to the world coordinate frame W.
${}_{W}\ddot{\mathbf{X}}_{R} \in \mathbb{R}^{6}$	Right Foot Acceleration	`CartesianAcceleration`	This vector contains the linear and angular accelerations of the right foot R with respect to the world coordinate frame W.
${}_{W}\ddot{\mathbf{X}}_{M} \in \mathbb{R}^{6}$	CoM Acceleration	`CartesianAcceleration`	This vector contains the linear and angular accelerations of the CoM with respect to the world coordinate frame W.

FK of the Commanded Robot

Symbol	Name	Type	Description
${}^{L_c}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Left Commanded Foot Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix transforms coordinates in the left commanded foot coordinate frame L to the world coordinate frame W.
${}^{R_c}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Right Commanded Foot Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix transforms coordinates in the right commanded foot coordinate frame R to the world coordinate frame W.
${}^{M_c}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	CoM Commanded Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix transforms coordinates in the CoM commanded coordinate frame M to the world coordinate frame W.
${}_{W}\dot{\mathbf{X}}_{L_c} \in \mathbb{R}^{6}$	Left Commanded Foot Velocity	`CartesianVelocity`	This vector contains the linear and angular velocities of the left commanded foot L with respect to the world coordinate frame W.
${}_{W}\dot{\mathbf{X}}_{R_c} \in \mathbb{R}^{6}$	Right Commanded Foot Velocity	`CartesianVelocity`	This vector contains the linear and angular velocities of the commanded right foot R with respect to the world coordinate frame W.
${}_{W}\dot{\mathbf{X}}_{M_c} \in \mathbb{R}^{6}$	CoM Commanded Velocity	`CartesianVelocity`	This vector contains the linear and angular velocities of the commanded CoM with respect to the world coordinate frame W.
${}_{W}\ddot{\mathbf{X}}_{L_c} \in \mathbb{R}^{6}$	Left Commanded Foot Acceleration	`CartesianAcceleration`	This vector contains the linear and angular accelerations of the commanded left foot L with respect to the world coordinate frame W.
${}_{W}\ddot{\mathbf{X}}_{R_c} \in \mathbb{R}^{6}$	Right Commanded Foot Acceleration	`CartesianAcceleration`	This vector contains the linear and angular accelerations of the commanded right foot R with respect to the world coordinate frame W.
${}_{W}\ddot{\mathbf{X}}_{M_c} \in \mathbb{R}^{6}$	CoM Commanded Acceleration	`CartesianAcceleration`	This vector contains the linear and angular accelerations of the commanded CoM with respect to the world coordinate frame W.

2.3 Inter-Connections

The inputs of the FKMs are connected to the outputs of the Real Robot Module (RRM) which provides the joint positions, velocities, and accelerations of the real and the commanded robot. The outputs of FKMs (real and commanded) are connected to modules that require the Cartesian positions, velocities, and accelerations of the left/right foot and the CoM end-effectors with respect to the world. The real FKM is connected to

- the Zero-Moment-Point Module (ZMPM),
- the Center-of-Mass Module (CoMM) for the real robot, and

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

- the Foot Compliance Model Module (FCMM).

The commanded FKM is connected to

- the Center-of-Mass Module (CoMM) for the commanded robot, and
- the Foot Trajectory Generator Module (FTGM).

The ZMPM module uses ${}^{\mathrm{L}}_{\mathrm{W}}\mathbf{T}$, ${}^{\mathrm{R}}_{\mathrm{W}}\mathbf{T}$, ${}^{\mathrm{M}}_{\mathrm{W}}\mathbf{T}$, and ${}_{\mathrm{W}}\dot{\mathbf{X}}_{\mathrm{M}}$ in combination with information of the foot FT sensors and the IMU to compute the linear position, and velocity, of the Zero-Moment-Point (ZMP). The CoMMs (real and commanded) fuse ${}^{\mathrm{M}}_{\mathrm{W}}\mathbf{T}$ and ${}_{\mathrm{W}}\dot{\mathbf{X}}_{\mathrm{M}}$ with IMU information in a model based filter to estimate the Cartesian position and velocity of the CoM and the linear position and velocity of the capture point (CP). The FCMM requires ${}^{\mathrm{L}}_{\mathrm{W}}\mathbf{T}$, ${}^{\mathrm{R}}_{\mathrm{W}}\mathbf{T}$, ${}_{\mathrm{W}}\dot{\mathbf{X}}_{\mathrm{L}}$, and ${}_{\mathrm{W}}\dot{\mathbf{X}}_{\mathrm{R}}$ compute the homogeneous transformation for the offset of the feet coordinate frames. The FTGM requires ${}^{\mathrm{L_c}}_{\mathrm{W}}\mathbf{T}$ and ${}^{\mathrm{R_c}}_{\mathrm{W}}\mathbf{T}$ to compute the reference Cartesian positions, velocities, and accelerations of the feet.

2.4 Common Methods

This module uses kinematic parameters such as joint properties (location, type), and link properties (location, length) to build up a rigid multi body system (MBS) that represents the kinematic model of the robot. The MBS is a tree of links and joints where the joints connect links. Relative spatial transformations between links and joints describe the spatial relation between parent and child links. Then the transformation of all coordinate frames within the MBS can be computed with respect to a reference coordinate frame by traveling along the branches of the tree and chaining up relative transformations between parent and child links. This recursive method of computing the transformations of link coordinate frames with respect to a reference coordinate frame has in contrast to the symbolic code generation method the advantage that existing models can be extended and more easily analyzed [1]. Furthermore, the recursive method does not require the complex generation of code from symbolic expressions.

References

[1] Martin L. Felis, RBDL: An efficient rigid-body dynamics library using recursive algorithms, Autonomous Robots 41 (2): 495–511, 2017.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Module Description: Zero Moment Point (ZMPM)

Emmanuel Dean, Florian Bergner, Rogelio Guadarrama-Olvera,
Simon Armleder, and Gordon Cheng

February 14, 2020

1 Module Description

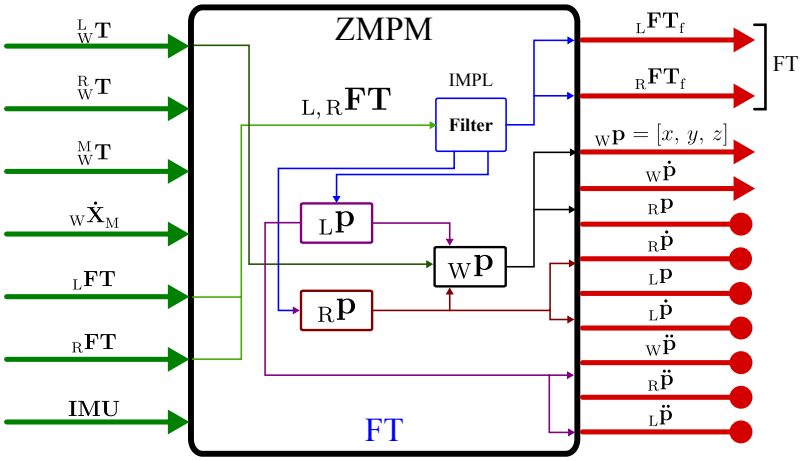

Figure 1.1: Zero Moment Point module: This module implements the local zmp estimation for each leg and the combined legs ${}_{\mathrm{W}}\mathbf{p}$, ${}_{\mathrm{L}}\mathbf{p}$, and ${}_{\mathrm{R}}\mathbf{p}$, respectively.

The *Zero Moment Point* module (ZMPM) estimates the local Zero Moment Points (ZMP) for each foot and the global ZMP for both feet, ${}_{\mathrm{W}}\mathbf{p}$, ${}_{\mathrm{L}}\mathbf{p}$, and ${}_{\mathrm{R}}\mathbf{p}$, respectively. The ZMP is an important concept for dynamics and control of legged locomotion, e.g., for humanoid robots. It specifies the point where the dynamic reaction forces at the contact of the foot with the ground does not produce any moment in the horizontal direction, i.e. the point where the total of horizontal inertia and gravity forces are in equilibrium. This module requires kinematic information of the feet, dynamic information of the Center of Mass (CoM), and ground reaction

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

forces, which can be obtained with Force/Torque (FT) sensors, for example, mounted on the feet. The ZMP calculation can be extended using IMU sensors as well. The information obtained from the ZMP analysis is extremely important for balance, which is the highest priority task for legged robots. This module also filters the signals of the FT sensors, which are used by other components of the OpenWalker framework.

2 Module Connections

2.1 Inputs

Symbol	Name	Type	Description
${}^{L}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Left Foot Pose	`HomogeneousTransformation`	This matrix represents the pose of the left foot with respect to the world coordinate frame (wcf).
${}^{R}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Right Foot Pose	`HomogeneousTransformation`	This matrix represents the pose of the right foot with respect to the world coordinate frame (wcf).
${}^{M}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Real Robot CoM Pose	`HomogeneousTransformation`	This matrix represents the pose of the real CoM with respect to the world coordinate frame (wcf).
${}_{W}\dot{\mathbf{x}}_{M} \in \mathbb{R}^{6}$	Real Robot CoM Velocities	`CartesianVelocity`	This vector contains the real CoM velocities of the robot.
${}_{L}\mathbf{FT} \in \mathbb{R}^{6}$	Left Foot FT	`ForceTorqueSensor`	This vector contains the signals of the force/torque sensor mounted on the left foot.
${}_{R}\mathbf{FT} \in \mathbb{R}^{6}$	Right Foot FT	`ForceTorqueSensor`	This vector contains the signals of the force/torque sensor mounted on the right foot.
$\mathbf{IMU} \in \mathbb{R}^{10}$	IMU information	`ImuSensor`	This vector contains the IMU sensor information which includes Cartesian acceleration, angular position (in Quaternions), and angular velocity of the hip/torso.

2.2 Outputs

Symbol	Name	Type	Description
${}_{L}\mathbf{FT}_{f} \in \mathbb{R}^{6}$	Filtered Left Foot FT	`ForceTorqueSensor`	This vector contains the signals of the filtered force/torque sensor mounted on the left foot.
${}_{R}\mathbf{FT}_{f} \in \mathbb{R}^{6}$	Filtered Right Foot FT	`ForceTorqueSensor`	This vector contains the signals of the filtered force/torque sensor mounted on the right foot.
${}_{W}\mathbf{p} \in \mathbb{R}^{3}$	Zero Moment Point	`ZeroMomentPoint`	This vector represents the combined zero moment point with respect to the world coordinate frame.
${}_{W}\dot{\mathbf{p}} \in \mathbb{R}^{3}$	Zero Moment Velocity	`ZeroMomentPointP`	This vector represents the time derivative of the combined zero moment point with respect to the world coordinate frame.
${}_{W}\ddot{\mathbf{p}} \in \mathbb{R}^{3}$	Zero Moment Acceleration	`ZeroMomentPointPP`	This vector represents the 2nd time derivative of the combined zero moment point with respect to the world coordinate frame.
${}_{L}\mathbf{p} \in \mathbb{R}^{3}$	Left Foot Zero Moment Point	`ZeroMomentPoint`	This vector represents the zero moment point with respect to the left foot coordinate frame.
${}_{L}\dot{\mathbf{p}} \in \mathbb{R}^{3}$	Left Foot Zero Moment Velocity	`ZeroMomentPointP`	This vector represents the time derivative of the zero moment point with respect to the left foot coordinate frame.
${}_{L}\ddot{\mathbf{p}} \in \mathbb{R}^{3}$	Left Foot Zero Moment Acceleration	`ZeroMomentPointPP`	This vector represents the 2nd time derivative of the zero moment point with respect to the left foot coordinate frame.
${}_{R}\mathbf{p} \in \mathbb{R}^{3}$	Right Foot Zero Moment Point	`ZeroMomentPoint`	This vector represents the zero moment point with respect to the right foot coordinate frame.
${}_{R}\dot{\mathbf{p}} \in \mathbb{R}^{3}$	Right Foot Zero Moment Velocity	`ZeroMomentPointP`	This vector represents the time derivative of the zero moment point with respect to the right foot coordinate frame.
${}_{R}\ddot{\mathbf{p}} \in \mathbb{R}^{3}$	Right Foot Zero Moment Acceleration	`ZeroMomentPointPP`	This vector represents the 2nd time derivative of the zero moment point with respect to the right foot coordinate frame.

2.3 Inter-Connections

The inputs of the ZMPM come from different sources. The RRM provides the FT sensors information and the IMU sensor information, ${}_{L}\mathbf{FT}$, ${}_{R}\mathbf{FT}$, $\mathbf{IMU}$, respectively. The poses of the

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

feet and the CoM with respect to the world coordinate frame $\left({}^{\mathrm{L}}_{\mathrm{W}}\mathbf{T}, {}^{\mathrm{R}}_{\mathrm{W}}\mathbf{T}, {}^{\mathrm{M}}_{\mathrm{W}}\mathbf{T}\right)$ as well as the CoM velocity $({}_{\mathrm{W}}\dot{\mathbf{x}}_{\mathrm{M}})$ are provided by the FKM, using the real robot joint states $\left(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}\right)$.

This ZMPM provides the information of the combined zmp $\left({}_{\mathrm{W}}\mathbf{p}\right)$ and its time-derivative $\left({}_{\mathrm{W}}\dot{\mathbf{p}}\right)$ for the balancer (BM), and the filtered FT sensor signals $\left({}_{\mathrm{L}}\mathbf{FT}_{\mathrm{f}}, {}_{\mathrm{L}}\mathbf{FT}_{\mathrm{f}}\right)$ to compute the compliance of the feet in the FCM.

2.4 Common Methods

2.4.1 ZMP

The standard method to compute the ZMP is based directly on the FT sensor information of each foot [1].

First, the ZMP of each foot should be calculated:

$$ {}_{\mathrm{F}}\mathbf{p}_{\mathrm{x}} = \frac{-{}_{\mathrm{F}}\boldsymbol{\mu}_{\mathrm{y}} - {}_{\mathrm{F}}\mathbf{f}_{\mathrm{x}}\, d_{\mathrm{F}}}{{}_{\mathrm{F}}\mathbf{f}_{\mathrm{z}}} \tag{2.1} $$

$$ {}_{\mathrm{F}}\mathbf{p}_{\mathrm{y}} = \frac{-{}_{\mathrm{F}}\boldsymbol{\mu}_{\mathrm{x}} - {}_{\mathrm{F}}\mathbf{f}_{\mathrm{y}}\, d_{\mathrm{F}}}{{}_{\mathrm{F}}\mathbf{f}_{\mathrm{z}}} \tag{2.2} $$

$$ {}_{\mathrm{F}}\mathbf{p} = \left[{}_{\mathrm{F}}\mathbf{p}_{\mathrm{x}}, {}_{\mathrm{F}}\mathbf{p}_{\mathrm{y}}, 0\right]^{\top} \tag{2.3} $$

where F = {L, R}, ${}_{\mathrm{F}}\mathbf{FT} = \left[{}_{\mathrm{F}}\mathbf{f}, {}_{\mathrm{F}}\boldsymbol{\mu}\right]$, and d_{F} is the FT sensor offset of the foot F.

The ZMP of both feet is combined and represented with respect to the world coordinate frame. To this aim, first, we project each ZMP to the world coordinate frame (wcf) using the orientation of each feet w.r.t the wcf, ${}^{\mathrm{F}}_{\mathrm{W}}\mathbf{R} \in SO(3)$, which is obtained from the feet pose ${}^{\mathrm{F}}_{\mathrm{W}}\mathbf{T} \in \mathbb{R}^{4\times4}$.

$$ {}_{\mathrm{W}}\mathbf{p}_{\mathrm{F}} = {}^{\mathrm{F}}_{\mathrm{W}}\mathbf{R}\, {}_{\mathrm{F}}\mathbf{p} \tag{2.4} $$

Finally, each projected ZMP, $\left({}_{\mathrm{W}}\mathbf{p}_{\mathrm{F}}\right)$ with F = {L, R}, is combined to produce the global ZMP w.r.t the wcf.

$$ {}_{\mathrm{W}}\mathbf{p}_{\mathrm{x}} = \frac{{}_{\mathrm{W}}\mathbf{p}_{\mathrm{R},x}\, {}_{\mathrm{W}}\mathbf{f}_{\mathrm{R},z} + {}_{\mathrm{W}}\mathbf{p}_{\mathrm{L},x}\, {}_{\mathrm{W}}\mathbf{f}_{\mathrm{L},z}}{{}_{\mathrm{W}}\mathbf{f}_{\mathrm{L},z} + {}_{\mathrm{W}}\mathbf{f}_{\mathrm{R},z}} \tag{2.5} $$

$$ {}_{\mathrm{W}}\mathbf{p}_{\mathrm{y}} = \frac{{}_{\mathrm{W}}\mathbf{p}_{\mathrm{R},y}\, {}_{\mathrm{W}}\mathbf{f}_{\mathrm{R},z} + {}_{\mathrm{W}}\mathbf{p}_{\mathrm{L},y}\, {}_{\mathrm{W}}\mathbf{f}_{\mathrm{L},z}}{{}_{\mathrm{W}}\mathbf{f}_{\mathrm{L},z} + {}_{\mathrm{W}}\mathbf{f}_{\mathrm{R},z}} \tag{2.6} $$

2.4.2 2nd Order Filtering

To filter the FT sensor signals, we can use a second order filter, for example, Butterworth filter [2]. This filter can be computed as:

$$ x_{\mathrm{f},k} = b_0 x_k + b_1 x_{k-1} + b_2 x_{k-2} - a_1 x_{\mathrm{f},k-1} - a_2 x_{\mathrm{f},k-2} \tag{2.7} $$

where a_i and b_j are the filter coefficients that depend on the cutoff frequency f_{c} as:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

$$b_0 = \frac{T^2\, w_c^2}{d} \tag{2.8}$$

$$b_1 = \frac{T^2\, 2\, w_c^2 0}{d} \tag{2.9}$$

$$b_2 = \frac{T^2\, w_c^2}{d} \tag{2.10}$$

$$a_0 = 1.0 \tag{2.11}$$

$$a_1 = \frac{T^2\, 2\, w_c^2 - 8}{d} \tag{2.12}$$

$$a_2 = \frac{T^2\, w_c^2 - 2\sqrt{2}\, T\, w_c + 4}{d} \tag{2.13}$$

References

[1] Englsberger, Johannes, et al. Three-dimensional bipedal walking control based on divergent component of motion, IEEE Transactions on Robotics, pp. 355-368, 2015.

[2] George Ellis. Filters in Control Systems, Chapter of Control System Design Guide (Fourth Edition), pp. 165-183, 2012.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Module Description: Center of Mass (CoMM)

Emmanuel Dean, Florian Bergner, Rogelio Guadarrama-Olvera,
Simon Armleder, and Gordon Cheng

February 14, 2020

1 Module Description

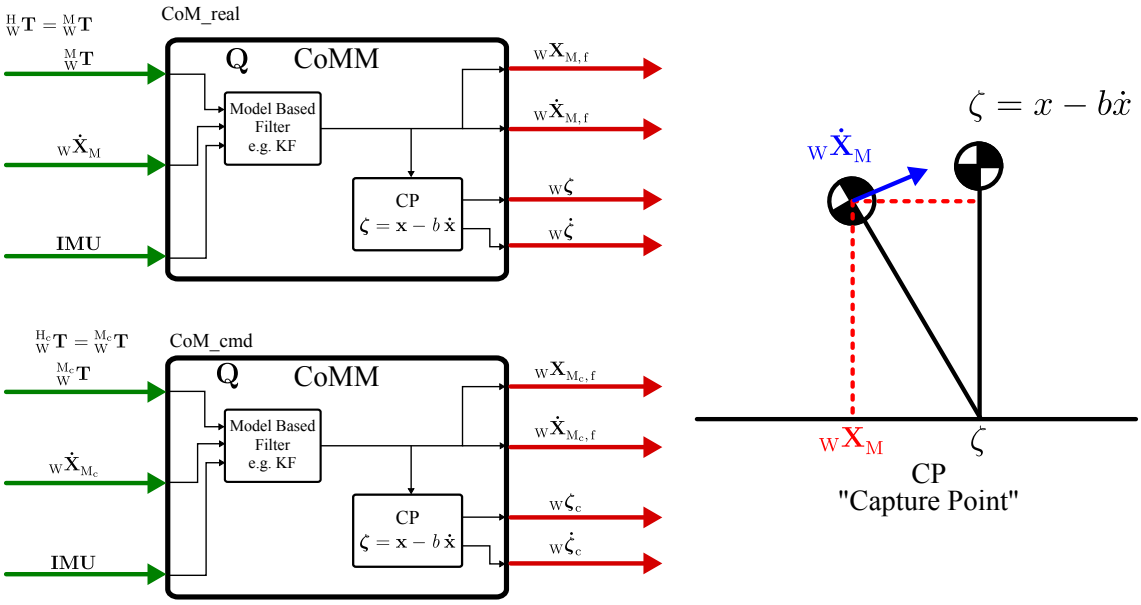

Figure 1.1: Center of Mass module: This module implements the center of mass estimation for the lower body of the robot, and computes the Capture point (CP) ${}_{\mathrm{W}}\boldsymbol{\zeta}$.

The *Center of Mass* module (CoMM) estimates the state of the *Center of Mass* (CoM) of the lower body of the robot using a model-based estimator, e.g. Kalman Filter [2]. This estimator uses as an input the Cartesian position and velocity of the CoM obtained from the joint sates

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

and the dynamic library (KDL/RBDL/KINDR), fused with the information from an IMU. The OpenWalker framework employs two CoMMs, one for computing the CoM of the real robot, and one for the commanded robot. The joint states of both robots can differ depending on the conditions of the environment.

The second task of this module is the real-time computation of the Divergent Component of Motion (DCM), also known as Capture Point (CP)[3]. The DCM is a major concept that has been applied to walking and running pattern generation. This CP most be computed for both the *real* and the *commanded* robot models. The CoM calculation of the *real* robot uses the Homogeneous transformation of the CoM *w.r.t* the *world* coordinate frame (wcf), ${}^{\mathrm{M}}_{\mathrm{W}}\mathbf{T}$, the Cartesian velocity of the CoM, ${}_{\mathrm{W}}\dot{\mathbf{x}}_{\mathrm{M}}$, and the IMU information[1]. Correspondingly, the CoM of the *commanded* robot uses the commanded CoM position and velocity, ${}^{\mathrm{M_c}}_{\mathrm{W}}\mathbf{T}$ and ${}_{\mathrm{W}}\dot{\mathbf{x}}_{\mathrm{M_c}}$, respectively. The OpenWalker framework uses the *real* and *commanded* CoM state in the Balancer (BM) to produce adequate hip offsets to keep the balance of the robot.

Since the rigid multi body system (MBS) of the robot is the same for the *real* and the *commanded* robot the OpenWalker framework needs to realize only one CoMM, which is then implemented and connected to the *real* Cartesian state, and to the *commanded* Cartesian state.

2 Module Connections

2.1 Inputs

Symbol	Name	Type	Description
${}^{\mathrm{M}}_{\mathrm{W}}\mathbf{T} \in \mathbb{R}^{4\times4}$	Real Robot CoM Pose	`HomogeneousTransformation`	This matrix represents the pose of the real CoM with respect to the world coordinate frame (wcf).
${}_{\mathrm{W}}\dot{\mathbf{x}}_{\mathrm{M}} \in \mathbb{R}^6$	Real Robot CoM Velocities	`CartesianVelocity`	This vector contains the real CoM velocities of the robot.
$\mathbf{IMU} \in \mathbb{R}^{10}$	IMU information	`ImuSensor`	This vector contains the IMU sensor information which includes Cartesian acceleration, angular position (in Quaternions), and angular velocity of the hip/torso.
${}^{\mathrm{M_c}}_{\mathrm{W}}\mathbf{T} \in \mathbb{R}^{4\times4}$	Commanded Robot CoM Pose	`HomogeneousTransformation`	This matrix represents the pose of the commanded CoM with respect to the world coordinate frame (wcf).
${}_{\mathrm{W}}\dot{\mathbf{x}}_{\mathrm{M_c}} \in \mathbb{R}^6$	Commanded Robot CoM Velocities	`CartesianVelocity`	This vector contains the commanded CoM velocities of the robot.
$\mathbf{IMU} \in \mathbb{R}^{10}$	IMU information	`ImuSensor`	This vector contains the IMU sensor information which includes Cartesian acceleration, angular position (in Quaternions), and angular velocity of the hip/torso.

2.2 Outputs

CoM of the Real Robot

Symbol	Name	Type	Description
${}_{\mathrm{W}}\mathbf{X}_{\mathrm{M,f}} \in \mathbb{R}^6$	Real robot filtered CoM pose	`CartesianPosition`	This vector represents the filtered pose (position and orientation) of the real CoM.
${}_{\mathrm{W}}\dot{\mathbf{X}}_{\mathrm{M,f}} \in \mathbb{R}^6$	Real robot filtered CoM velocity	`CartesianVelocity`	This vector represents the filtered Cartesian velocity (linear and angular) of the real CoM.
${}_{\mathrm{W}}\boldsymbol{\zeta} \in \mathbb{R}^3$	Real robot 3D Capture Point	`DivergentComponentOfMotion`	This vector represents the DCM position of the real robot.
${}_{\mathrm{W}}\dot{\boldsymbol{\zeta}} \in \mathbb{R}^3$	Real robot 3D Capture Point velocity	`DivergentComponentOfMotionP`	This vector represents the DCM velocity of the real robot.

[1] Usually the IMU sensor is located either in the hip or the torso of the robot.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

CoM of the Commanded Robot

Symbol	Name	Type	Description
${}_{W}\mathbf{X}_{M_c,f} \in \mathbb{R}^6$	Commanded Robot filtered CoM pose	`CartesianPosition`	This vector represents the filtered pose (position and orientation) of the commanded CoM.
${}_{W}\dot{\mathbf{X}}_{M_c,f} \in \mathbb{R}^6$	Commanded Robot filtered CoM velocity	`CartesianVelocity`	This vector represents the filtered Cartesian velocity (linear and angular) of the commanded CoM.
${}_{W}\boldsymbol{\zeta}_c \in \mathbb{R}^3$	Commanded robot 3D Capture Point	`DivergentComponentOfMotion`	This vector represents the DCM position of the commanded robot.
${}_{W}\dot{\boldsymbol{\zeta}}_c \in \mathbb{R}^3$	Commanded robot 3D Capture Point velocity	`DivergentComponentOfMotionP`	This vector represents the DCM velocity of the commanded robot.

2.3 Inter-Connections

The inputs of the CoMM are connected to the outputs of the Forward Kinematic Model (FKM) which provides the pose and velocity of the CoM of the *real* and *commanded* robot. The CoMM input also is connected to the Real Robot Module (RRM) which provides a hardware interface for the IMU sensor. The outputs of CoMM (*real* and *commanded*) are directly connected to the Balancer module (BM).

2.4 Common Methods

The two main tasks of this module is the CoM state estimation and the computation of the DCM.

2.4.1 CoM Estimation

The general description of the state estimation is defined by [2]. This can be summarized as: Prediction of the state estimate:

$$\hat{\mathbf{X}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{X}}_{k|k-1} + \mathbf{B}_k \mathbf{u}_k \tag{2.1}$$

Prediction of the estimate covariance:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\top + \mathbf{Q}_k \tag{2.2}$$

Innovation of the residual:

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \tag{2.3}$$

Innovation of the covariance:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k \tag{2.4}$$

Optimal Kalman gain:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \mathbf{S}_k^{-1} \tag{2.5}$$

update the state estimate

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{2.6}$$

update of covariance:

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\, \mathbf{P}_{k|k-1} \tag{2.7}$$

with the updated residual as:

$$\tilde{\mathbf{y}}_{k|k} = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k} \tag{2.8}$$

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

2.4.2 DCM

The CP is a characteristic point of the linear inverted pendulum model. It was designed to address a question of push recovery: where should the robot step (instantaneously) to eliminate linear momentum ${}_{\mathrm{m}}\dot{\mathbf{p}}_{\mathrm{G}}$ and come (asymptotically) to a stop (See Fig. 1.1)? The general equation to compute the DCM is given by:

$$ {}_{\mathrm{W}}\boldsymbol{\zeta} = {}_{\mathrm{W}}\mathbf{x}_{\mathrm{M}} - b\,{}_{\mathrm{W}}\dot{\mathbf{x}}_{\mathrm{M}} \tag{2.9} $$

where ${}_{\mathrm{W}}\boldsymbol{\zeta} = {}_{\mathrm{W}}\left[\boldsymbol{\zeta}_{\mathrm{x}}, \boldsymbol{\zeta}_{\mathrm{y}}, \boldsymbol{\zeta}_{\mathrm{z}}\right]^{\top}$ is the DCM, ${}_{\mathrm{W}}\mathbf{x}_{\mathrm{M}} = {}_{\mathrm{W}}\left[x, y, z\right]^{\top}$ and ${}_{\mathrm{W}}\dot{\mathbf{x}}_{\mathrm{M}} = {}_{\mathrm{W}}\left[\dot{x}, \dot{y}, \dot{z}\right]^{\top}$ are the CoM position and velocity and $b > 0$ is the time-constant of the DCM dynamics [3].

References

[1] Martin L. Felis, RBDL: An efficient rigid-body dynamics library using recursive algorithms, Autonomous Robots 41 (2): 495–511, 2017.

[2] R. E. Kalman, A New Approach to Linear Filtering and Prediction Problems, Transaction of the ASME–Journal of Basic Engineering, pp. 35-45, March 1960.

[3] Englsberger, Johannes, et al. Three-dimensional bipedal walking control based on divergent component of motion, Ieee transactions on robotics, pp. 355-368, 2015.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Module Description: DCM Planner (DCMPSM)

Rogelio Guadarrama-Olvera, Emmanuel Dean, Florian Bergner,
Simon Armleder, and Gordon Cheng

February 14, 2020

1 Module Description

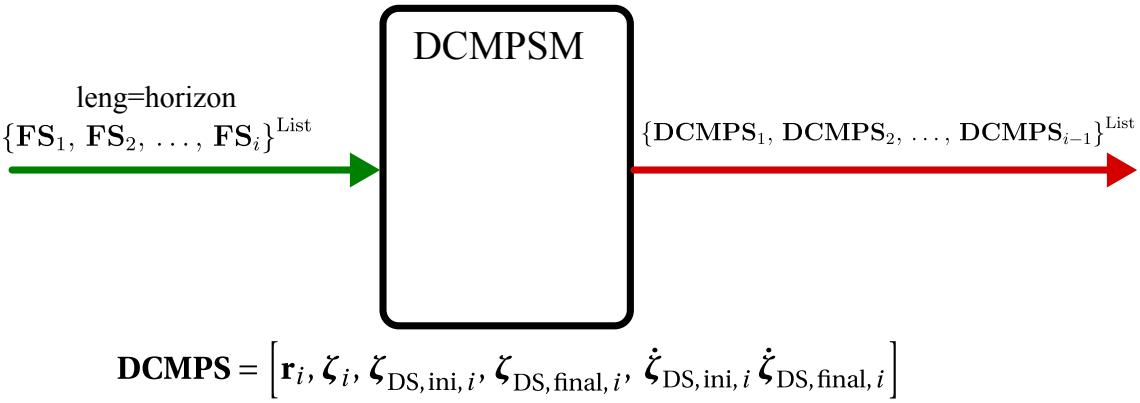

Figure 1.1: DCM Planner module: This module generates a set of way points for the Divergent Component of Motion (also known as Capture Point) controller, from the planned footsteps.

This module receives the planned feasible footsteps and computes the way points for the Divergent Component of Motion (DCM) dynamics for a stable walking motion. These points describe the transitions between one footstep to the next for the DCM which define an attractor point for the Center of Mass dynamics using the Linear Inverted Pendulum Model. These points also include the initial and final point of the double support phase used to smooth discontinuities on the DCM trajectory.

The computed **DCM** points for the i-th step ($\mathbf{DCMPS}_i$) are the Virtual Repellent Point $\mathbf{r}_i \in \mathbb{R}^3$, the step Capture Point $\boldsymbol{\zeta}_i \in \mathbb{R}^3$, the initial Capture Point for the double support phase

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

$\boldsymbol{\zeta}_{\text{DS,ini},i} \in \mathbb{R}^3$, the final Capture Point for the double support phase $\boldsymbol{\zeta}_{\text{DS,final},i} \in \mathbb{R}^3$, the initial Capture Point velocity for the double support phase $\dot{\boldsymbol{\zeta}}_{\text{DS,ini},i} \in \mathbb{R}^3$, the final Capture Point velocity for the double support phase $\dot{\boldsymbol{\zeta}}_{\text{DS,final},i} \in \mathbb{R}^3$.

This module must update the planned way points every time that the footstep plan is changed. However, the computation of these points is not very demanding for a reduced number of steps ahead. The algorithms to compute DCM way points require a minimum plan of 4 steps ahead of the current executed step.

2 Module Connections

2.1 Inputs

Symbol	Name	Type	Description
$\{\mathbf{FS}_i\}$	Footstep List	`FootStepList`	A list containing the planned footsteps, where $\mathbf{FS}_i = \left[\mathbf{X}_{\text{FS},i},\, l_i,\, s_i,\, n_i\right]$ with $\mathbf{X}_{\text{FS},i} = \left[\mathbf{x}_{\text{FS},i},\, \mathbf{Q}_{\text{FS},i}\right]$

2.2 Outputs

Symbol	Name	Type	Description
$\{\mathbf{DCMPS}_i\}$	DCM Point Set List	`DCMPointSet`	A list containing the DCM way points needed to execute the planned footsteps, where $\mathbf{DCMPS}_i = \left[\mathbf{r}_i,\, \boldsymbol{\zeta}_i,\, \boldsymbol{\zeta}_{\text{DS,ini},i},\, \boldsymbol{\zeta}_{\text{DS,final},i},\, \dot{\boldsymbol{\zeta}}_{\text{DS,ini},i},\, \dot{\boldsymbol{\zeta}}_{\text{DS,final},i}\right]$

2.3 Inter-Connections

This module receives the list of planned footsteps from the Footstep Planner (FSP) module. This port is shared with the Foot Trajectory Generator (FTG) module. The output of the DCMP module is connected to the CoM Trajectory Generator module.

2.4 Common Methods

There are different ways of generating stable walking motions. The most simple method considers an instantaneous double support phase and instantaneous foot transitions [1]. The process starts with a set of n foot locations from the footstep plan. Then, for the i-th foot location $\mathbf{x}_{\text{FS},i}$, a virtual repellent point is defined as

$$\mathbf{r}_i = \mathbf{x}_{\text{FS},i} + \begin{bmatrix} 0 & 0 & z_{\text{M}} \end{bmatrix}^\top \tag{2.1}$$

where z_{M} is the defined height for the CoM. Then, the i-th way point for the DCM can be computed as

$$\boldsymbol{\zeta}_i = \mathbf{r}_{i+1} + \mathrm{e}^{-\omega t_{\text{step}}} \left(\boldsymbol{\zeta}_{i+1} - \mathbf{r}_{i+1}\right) \tag{2.2}$$

$$\dot{\boldsymbol{\zeta}}_i = -\omega\, t_{\text{step}}\, \mathrm{e}^{-\omega t_{\text{step}}} \left(\boldsymbol{\zeta}_{i+1} - \mathbf{r}_{i+1}\right) \tag{2.3}$$

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

where $\omega = \sqrt{\frac{g}{z_M}}$ is the natural frequency of the LIPM and t_{step} is the step time defined in the walking parameters.

The minor index in the left side of (2.3) than in the right side means that these points must be computed from the last footstep in the plan to the current step. This also requires that $\boldsymbol{\zeta}_n = \boldsymbol{r}_n$ (the last planned step).

The instantaneous foot transitions result in discontinuous DCM trajectories which produce high tangential ground reaction forces on the standing foot. This condition is prone to foot skidding. To reduce these effects, one strategy to smooth the DCM trajectories is to introduce a continuous transition in a double support phase [3]. To achieve this, an initial and final way point for the double supporting phase must be computed as

$$\boldsymbol{\zeta}_{\text{DS,ini},i} = \mathbf{r}_{i-1} + e^{-0.5\omega t_{\text{DS}}}\left[\boldsymbol{\zeta}_i - \mathbf{r}_{i-1}\right] \tag{2.4}$$

$$\dot{\boldsymbol{\zeta}}_{\text{DS,ini},i} = -\omega\, t_{\text{step}}\, e^{-0.5\omega t_{\text{DS}}}\left[\boldsymbol{\zeta}_i - \mathbf{r}_{i-1}\right] \tag{2.5}$$

$$\boldsymbol{\zeta}_{\text{DS,final},i} = \mathbf{r}_i + e^{0.5\omega t_{\text{DS}}}\left[\boldsymbol{\zeta}_i - \mathbf{r}_i\right] \tag{2.6}$$

$$\dot{\boldsymbol{\zeta}}_{\text{DS,final},i} = -\omega\, t_{\text{step}}\, e^{0.5\omega t_{\text{DS}}}\left[\boldsymbol{\zeta}_i - \mathbf{r}_i\right] \tag{2.7}$$

where t_{DS} is the double support time defined in the walking parameters. A generalization of these methods for uneven terrain is described in [3].

References

[1] Englsberger, Johannes, et al. "Bipedal walking control based on capture point dynamics." 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2011.

[2] Romualdi, Giulio, et al. "A benchmarking of DCM based architectures for position and velocity controlled walking of humanoid robots." 2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids). IEEE, 2018.

[3] Englsberger, Johannes, et al. "Three-dimensional bipedal walking control based on divergent component of motion." Ieee transactions on robotics 31.2 (2015): 355-368.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Module Description: Foot Trajectory Generator (FTGM)

Simon Armleder, Emmanuel Dean, Florian Bergner,
Rogelio Guadarrama-Olvera, and Gordon Cheng

February 14, 2020

1 Module Description

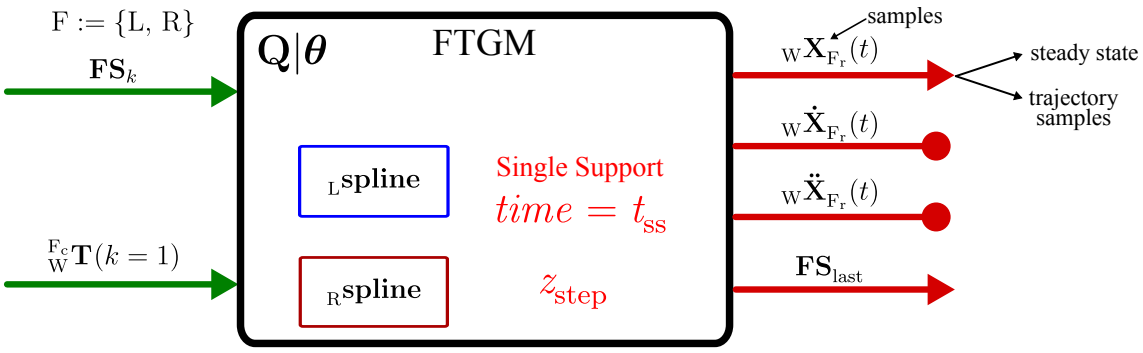

Figure 1.1: Foot Trajectory Generator: This module generates reference trajectories for the single-support phase.

The *Foot Trajectory Generator* module (FTG) constructs minimum jerk trajectories for the feet. These foot trajectories are then tracked by a controller during the single-support phase of the gait cycle. Generating smooth continuous trajectories is achieved by interpolating between the current foot pose and the next support foot pose.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

2 Module Connections

2.1 Inputs

Symbol	Name	Type	Description
${}^{L_r}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Left Foot Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix transforms coordinates in the left foot ankle coordinate frame L to the world coordinate frame W.
${}^{R_r}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Right Foot Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix transforms coordinates in the right foot ankle coordinate frame R to the world coordinate frame W.
$\mathbf{FS}_{\text{last}}$	Next Footstep	`FootStep`	The end position of the next footstep that gets executed.

2.2 Outputs

Symbol	Name	Type	Description
${}_{W}\mathbf{X}_{L_r} \in \mathbb{R}^7$	Left Foot Pose	`CartesianPosition`	This vector contains the linear and angular reference position of the left foot ankle L with respect to the world coordinate frame W.
${}_{W}\mathbf{X}_{R_r} \in \mathbb{R}^7$	Right Foot Pose	`CartesianPosition`	This vector contains the linear and angular reference position of the right foot ankle R with respect to the world coordinate frame W.
${}_{W}\dot{\mathbf{X}}_{L_r} \in \mathbb{R}^6$	Left Foot Velocity	`CartesianVelocity`	This vector contains the linear and angular reference velocities of the left foot ankle L with respect to the world coordinate frame W.
${}_{W}\dot{\mathbf{X}}_{R_r} \in \mathbb{R}^6$	Right Foot Velocity	`CartesianVelocity`	This vector contains the linear and angular reference velocities of the right foot ankle L with respect to the world coordinate frame W.
${}_{W}\ddot{\mathbf{X}}_{L_r} \in \mathbb{R}^6$	Left Foot Acceleration	`CartesianAcceleration`	This vector contains the linear and angular accelerations of the left foot ankle L with respect to the world coordinate frame W.
${}_{W}\ddot{\mathbf{X}}_{R_r} \in \mathbb{R}^6$	Right Foot Acceleration	`CartesianAcceleration`	This vector contains the linear and angular accelerations of the right foot ankle R with respect to the world coordinate frame W.

2.3 Inter-Connections

The first input of the FTG is connected to the output of the *Foot Step Planner* (FSP) which provides information about the next foot step to be executed. The second input of the FTG is connected to the *Forward Kinematics Module* (FKM). This connections provides the current pose of the left and right foot ankle with respect to the world coordinate frame.

The outputs of the FTG contain the interpolated feet trajectories tracked during the single-support phase. This includes the reference foot ankle position, velocity and acceleration. The FTG output is connected to the *Foot Compliant Module* (FKM) and *Command Generator* (CG).

2.4 Common Methods

Several algorithms exist to construct minimal jerk foot trajectories. Traditionally, foot trajectories are generated by polynomial interpolation with start and end boundary conditions of zero velocities and accelerations.
When there are various foot constrains such as ground conditions or obstacles, the order of these polynomials is too high and may oscillate. This problem can be avoided by utilizing cubic spline interpolation [1]. With this method foot trajectories are constructed of piecewise third-order polynomials which pass though a set of control points at chosen velocities. An advantage of spline trajectories is that intermediate control points can be easily shifted to deal with various obstacles during the stepping motion.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

References

[1] Verrelst, Bjorn and Stasse, Olivier and Yokoi, Kazuhito and Vanderborght, Bram, Dynamically stepping over obstacles by the humanoid robot HRP-2. 6th IEEE-RAS International Conference on Humanoid Robots, 2006.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Module Description: CoM Trajectory Generator (CoMTGM)

Rogelio Guadarrama-Olvera, Emmanuel Dean, Florian Bergner,
Simon Armleder, and Gordon Cheng

February 14, 2020

1 Module Description

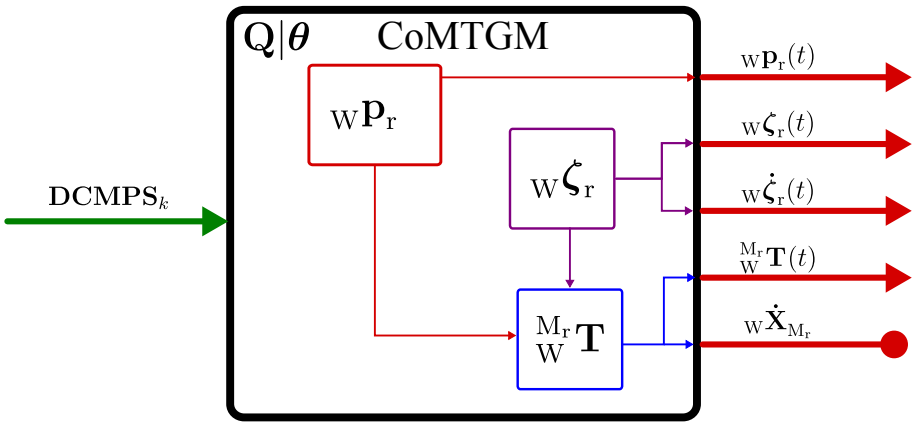

Figure 1.1: CoM Trajectory Generator module: This module generates a smooth reference trajectory for the center of mass by resolving the Divergent Component of Motion Dynamics.

This module receives the planned set of Divergent Component of Motion (DCM) way points and interpolates a smooth trajectory for both the DCM and the Center of Mass (CoM). The trajectories are generated following the DCM dynamics of the Linear Inverted Pendulum Model (LIPM). This module shifts the current way point to the next every time a step is finished, keeping the transition from one to the other continuous and smooth. The DCM moves

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

between the Virtual Repellent Points (VRP) and the CoM follows it with the natural dynamics of the LIPM. While the CoM is a point in space, a common practice to consider whole-body angular momentum is to consider a virtual rigid body at the CoM with the same orientation of the base link (commonly the hip or the torso).

The computed trajectories are the reference Zero Moment Point with respect to the world ${}_{W}\mathbf{p}_{r} \in \mathbb{R}^3$, the reference DCM and its derivative with respect to the world ${}_{W}\boldsymbol{\zeta}_{r}$, ${}_{W}\dot{\boldsymbol{\zeta}}_{r} \in \mathbb{R}^3$, the reference CoM position and the orientation of the virtual link as an homogeneous transformation with respect to the world ${}^{M_r}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$, the velocity of the virtual link ${}_{W}\dot{\mathbf{X}}_{M_r} \in \mathbb{R}^6$.

2 Module Connections

2.1 Inputs

Symbol	Name	Type	Description
$\{\mathbf{DCMPS}_i\}$	DCP Point Set List	`DCMPointSetList`	A list containing the DCM way points needed to execute the planned footsteps.

2.2 Outputs

Symbol	Name	Type	Description
${}_{W}\mathbf{p}_{r} \in \mathbb{R}^3$	Reference ZMP	`ZeroMomentPoint`	Reference trajectory for the ZMP.
${}_{W}\boldsymbol{\zeta}_{r} \in \mathbb{R}^3$	Reference DCM Position	`DivergentComponentOfMotion`	Reference trajectory for the DCM.
${}_{W}\dot{\boldsymbol{\zeta}}_{r} \in \mathbb{R}^3$	Reference DCM Velocity	`DivergentComponentOfMotionP`	Derivative of the reference trajectory for the DCM.
${}^{M_r}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Reference CoM Position	`HomogeneousTransformation`	Reference trajectory for the CoM. This includes both position and orientation of a virtual link located at the CoM with the same oriantation of the base link.
${}_{W}\dot{\mathbf{X}}_{M_r} \in \mathbb{R}^6$	Reference CoM Velocity	`CartesianVelocity`	Derivative of the reference trajectory for the CoM. This includes both linear and angular velocities of the base link.

2.3 Inter-Connections

This module receives the list of planned DCM waypoints from the DCM planner module. The outputs of this module is connected to the Balancer Module and the Command Generator Module. The balance module uses all the outputs while the Command Generator module uses only ${}^{M_r}_{W}\mathbf{T}$.

2.4 Common Methods

The dynamics of the LIPM are described by

$$\ddot{\mathbf{x}} = \omega^2 \left(\mathbf{x} - \mathbf{p}\right) \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^3$ is the position of the CoM and $\ddot{\mathbf{x}} \in \mathbb{R}^3$ its second derivative, $\mathbf{p} \in \mathbb{R}^3$ is the position of the ZMP, and $\omega = \sqrt{\frac{g}{z}}$ is the pendulum parameter which define its natural frequency.

The DCM is the result of a change of variable of the LIPM equation which is defined as

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

$$\boldsymbol{\zeta} = \mathbf{x} + \frac{\dot{\mathbf{x}}}{\omega} \tag{2.2}$$

$$\dot{\boldsymbol{\zeta}} = \dot{\mathbf{x}} + \frac{\ddot{\mathbf{x}}}{\omega} \tag{2.3}$$

with this representation, the equations 2.1 and 2.3 can be combined as

$$\dot{\mathbf{x}} = -\omega\left(\mathbf{x} - \boldsymbol{\zeta}\right) \tag{2.4}$$

$$\dot{\boldsymbol{\zeta}} = \omega\left(\boldsymbol{\zeta} - \mathbf{p}\right) \tag{2.5}$$

From (2.4) it is clear that the CoM follows the DCM in a first order stable dynamic system. However the dynamics of the DCM (2.5) are unstable. Nevertheless, these dynamics can be used to generate a stable reference trajectory for walking the i-th step by solving (2.5) for a constant $\mathbf{p} = \mathbf{r}_i$.

$${}_{\mathrm{W}}\boldsymbol{\zeta}_{\mathrm{r}} = \mathbf{r}_i + \mathrm{e}^{\omega\left(t - t_{\mathrm{step}}\right)}\left(\boldsymbol{\zeta}_i - \mathbf{r}_i\right) \tag{2.6}$$

where $\mathbf{r}_i$ is the i-th Virtual Repellent Point (VRP) and $\boldsymbol{\zeta}_i$ is the i-th DCM point from the way points planned by the DCMP module.

Finally, (2.4) can be numerically solved using (2.6) as

$${}_{\mathrm{W}}\mathbf{x}_{\mathrm{M_r},\,i+1} = {}_{\mathrm{W}}\boldsymbol{\zeta}_{\mathrm{r}} + \mathrm{e}^{-\omega t_\Delta}\left({}_{\mathrm{W}}\mathbf{x}_{\mathrm{M_r},\,i} - {}_{\mathrm{W}}\boldsymbol{\zeta}_{\mathrm{r}}\right) \tag{2.7}$$

where t_Δ is the iteration period.

References

[1] Englsberger, Johannes, et al. "Bipedal walking control based on capture point dynamics." 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2011.

[2] Englsberger, Johannes, et al. "Three-dimensional bipedal walking control based on divergent component of motion." Ieee transactions on robotics 31.2 (2015): 355-368.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Module Description: Foot Compliant (FCM)

Emmanuel Dean, Florian Bergner, Rogelio Guadarrama-Olvera,
Simon Armleder, and Gordon Cheng

February 14, 2020

1 Module Description

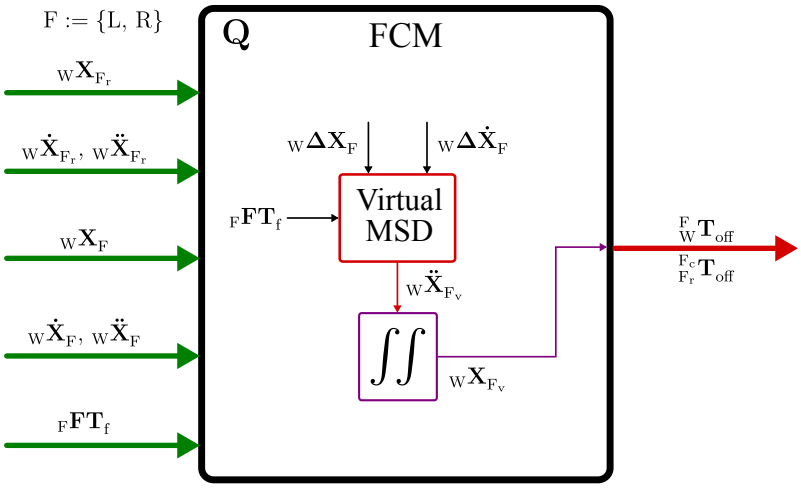

Figure 1.1: Foot Compliant module: This module implements the local zmp estimation for each leg and the combined legs ${}_{W}\mathbf{p}$, ${}_{L}\mathbf{p}$, and ${}_{R}\mathbf{p}$, respectively.

The *Foot Compliant* module (FCM) allows to modify the foot trajectories based on the conditions of the environment. The principal idea behind this module is to provide a compliant behavior for the foot to cope with uncertainties in the terrain. This compliant behavior uses the FT sensor information to generate offsets for the foot, which modify the nominal foot trajectory generated by the FTGM. The compliant behavior is generated using a virtual admittance model that is defined in the local foot coordinate frame. Admittance control is a standard control method to map forces to motion using simple virtual mass-damper dynamic systems.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

estimates the local Zero Moment Points (ZMP) for each foot and the global ZMP for both feet, ${}_{W}\mathbf{p}$, ${}_{L}\mathbf{p}$, and ${}_{R}\mathbf{p}$, respectively. The ZMP is an important concept for dynamics and control of legged locomotion, e.g., for humanoid robots. It specifies the point where the dynamic reaction forces at the contact of the foot with the ground does not produce any moment in the horizontal direction, i.e. the point where the total of horizontal inertia and gravity forces are in equilibrium. This module requires kinematic information of the feet, dynamic information of the Center of Mass (CoM), and ground reaction forces, which can be obtained with Force/Torque (FT) sensors, for example, mounted on the feet. The ZMP calculation can be extended using IMU sensors as well. The information obtained from the ZMP analysis is extremely important for balance, which is the highest priority task for legged robots. This module also filters the signals of the FT sensors, which are used by other components of the OpenWalker framework.

2 Module Connections

2.1 Inputs

Symbol	Name	Type	Description
${}_{W}\mathbf{X}_{F} \in \mathbb{R}^7$	Foot Pose	`CartesianPosition`	This vector represents the pose of the left and right feet with respect to the world coordinate frame (wcf). The orientation is represented as Quaternions.
${}_{W}\dot{\mathbf{X}}_{F} \in \mathbb{R}^6$	Foot Velocity	`CartesianVelocity`	This vector represents the velocity of the foot with respect to the world coordinate frame.
${}_{W}\ddot{\mathbf{X}}_{F} \in \mathbb{R}^6$	Foot Acceleration	`CartesianAcceleration`	This vector represents the acceleration of the foot with respect to the world coordinate frame.
${}_{W}\mathbf{X}_{F_r} \in \mathbb{R}^7$	Reference Foot Pose	`CartesianPosition`	This vector represents the reference pose of the left and right feet with respect to the world coordinate frame (wcf). The orientation is represented as Quaternions. This reference pose is obtained from a trajectory generator.
${}_{W}\dot{\mathbf{X}}_{F_r} \in \mathbb{R}^6$	Reference Foot Velocity	`CartesianVelocity`	This vector represents the reference velocity of the foot with respect to the world coordinate frame.This reference pose is obtained from a trajectory generator.
${}_{W}\ddot{\mathbf{X}}_{F_r} \in \mathbb{R}^6$	Reference Foot Acceleration	`CartesianAcceleration`	This vector represents the reference acceleration of the foot with respect to the world coordinate frame.This reference pose is obtained from a trajectory generator.
${}_{L}\mathbf{FT} \in \mathbb{R}^6$	Left Foot FT	`ForceTorqueSensor`	This vector contains the signals of the force/torque sensor mounted on the left foot.
${}_{R}\mathbf{FT} \in \mathbb{R}^6$	Right Foot FT	`ForceTorqueSensor`	This vector contains the signals of the force/torque sensor mounted on the right foot.

2.2 Outputs

Symbol	Name	Type	Description
${}^{F}_{W}\mathbf{T}_{off} \in \mathbb{R}^{4\times4}$	Global Foot Offset	`HomogeneousTransformation`	This homogeneous transformation provides the offset of the feet relative to the world coordinate frame.
${}^{F_c}_{F_r}\mathbf{T}_{off} \in \mathbb{R}^{4\times4}$	Local Foot Offset	`HomogeneousTransformation`	This homogeneous transformation provides the offset of the commanded feet relative to the reference foot coordinate frame.

2.3 Inter-Connections

The inputs of the FCM come from the FTGM which provides reference trajectories for the feet, $\left({}_{W}\mathbf{X}_{F_r}, {}_{W}\dot{\mathbf{X}}_{F_r}, {}_{W}\ddot{\mathbf{X}}_{F_r}\right)$, where F = {L, R}. The FKM produce the current foot state $\left({}_{W}\mathbf{X}_{F}, {}_{W}\dot{\mathbf{X}}_{F}, {}_{W}\ddot{\mathbf{X}}_{F}\right)$. Finally, the ZMPM provides the filtered FT sensor signals $\left({}_{F}\mathbf{FT}_{f}\right)$ needed to compute the offsets.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

The generated offset will be used in CMDGENM to compute a commanded trajectory for the feet.

2.4 Common Methods

2.4.1 Admittance based on Virtual Dynamics

The standard admittance method is based on a virtual mass-damping system, defined as [1]:

$$\mathbf{M}\,{}_{\mathrm{W}}\ddot{\mathbf{X}}_{\mathrm{v,F}} + \boldsymbol{\beta}\,{}_{\mathrm{W}}\dot{\mathbf{X}}_{\mathrm{v,F}} = {}_{\mathrm{W}}\mathbf{FT}_{\mathrm{F}} + {}_{\mathrm{W}}\mathbf{W}_{\delta,\mathrm{F}} \tag{2.1}$$

where $\mathbf{M} = \mathbf{M}^{\top}$ is a mass matrix, $\boldsymbol{\beta}$ is the viscous friction diagonal matrix, and ${}_{\mathrm{W}}\mathbf{FT}_{\mathrm{F}}$ is the FT sensor signals of the foot F, relative to the world coordinate frame. ${}_{\mathrm{F}}\mathbf{W}_{\delta}$ is a virtual attractor wrench generated with the error between the current foot pose and the reference foot pose, i.e.

$${}_{\mathrm{W}}\Delta\mathbf{X}_{\mathrm{F}} = {}_{\mathrm{W}}\mathbf{X}_{\mathrm{F}} - {}_{\mathrm{W}}\mathbf{X}_{\mathrm{F_r}} \tag{2.2}$$

This second order system can be integrated to generate virtual velocities and positions, ${}_{\mathrm{W}}\dot{\mathbf{X}}_{\mathrm{F_v}}$ and ${}_{\mathrm{W}}\mathbf{X}_{\mathrm{F_v}}$ respectively. Finally, the offset between the reference frame pose

$${}_{\mathrm{W}}\Delta\mathbf{X}_{\mathrm{off}} = {}_{\mathrm{W}}\mathbf{X}_{\mathrm{F_r}} - {}_{\mathrm{W}}\mathbf{X}_{\mathrm{F_v}} \tag{2.3}$$

is used to generate the pose matrices ${}^{\mathrm{F}}_{\mathrm{W}}\mathbf{T}_{\mathrm{off}}$ and ${}^{\mathrm{F_c}}_{\mathrm{F_r}}\mathbf{T}_{\mathrm{off}}$.

References

[1] Keemink, A. Q., van der Kooij, H., Stienen, A. H. Admittance control for physical human–robot interaction. The International Journal of Robotics Research, 37(11), 1421–1444, (2018).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Module Description: Balancer (BM)

Rogelio Guadarrama-Olvera, Emmanuel Dean, Florian Bergner,
Simon Armleder, and Gordon Cheng

February 14, 2020

1 Module Description

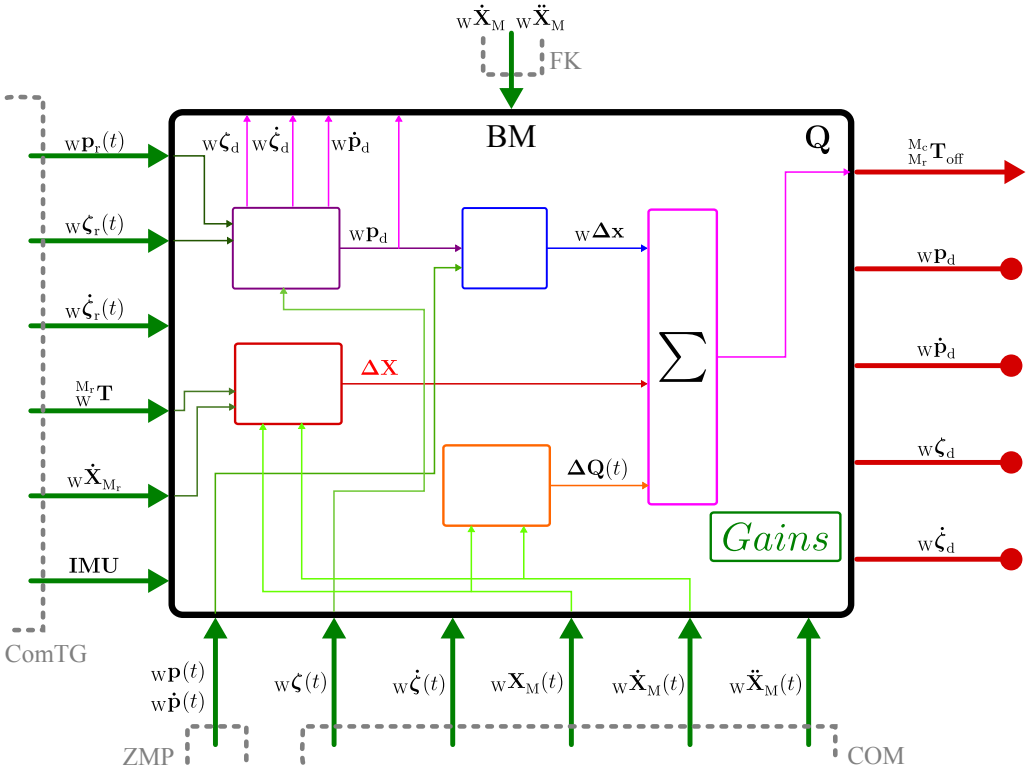

Figure 1.1: Balancer module: This module ensures the tracking of the reference CoM trajectory while suppressing external disturbances to keep both standing and walking balance.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

This module implements controllers to enforce the convergence of the real walking states (DCM, ZMP and CoM) to the reference trajectories. The desired states are computed from the reference trajectories in the form of continuous offset which are later added to the reference on the command generator module.

The module is composed by four controllers:

- **DCM tracker:** Computes a desired ZMP from the reference ZMP and the DCM error.
- **ZMP tracker:** Computes an offset to pull the CoM trajectory in order to track the desired ZMP with the real ZMP.
- **CoM Tracker:** Computes an offset to pull the commanded CoM trajectory to follow the reference CoM trajectory.
- **IMU orientation controller:** Computes a rotational offset fo correct the attitude of the base link to the reference orientation.

All the outputs of these controllers are then combined in an offset homogeneous transformation which will be used in the Command Generator module to compute the Command poses.

2 Module Connections

2.1 Inputs

Symbol	Name	Type	Description
${}_{W}\mathbf{p}_{r} \in \mathbb{R}^3$	Reference ZMP	`ZeroMomentPoint`	Reference trajectory for the ZMP.
${}_{W}\boldsymbol{\zeta}_{r} \in \mathbb{R}^3$	Reference DCM Position	`DivergentComponentOfMotion`	Reference trajectory for the DCM.
${}_{W}\dot{\boldsymbol{\zeta}}_{r} \in \mathbb{R}^3$	Reference DCM Velocity	`DivergentComponentOfMotion`	Derivative of the reference trajectory for the DCM.
${}^{M_r}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Reference CoM Position	`HomogeneousTransformation`	Reference trajectory for the CoM. This includes both position and orientation of a virtual link located at the CoM with the same oriantation of the base link.
${}_{W}\dot{\mathbf{X}}_{M_r} \in \mathbb{R}^6$	Reference CoM Velocity	`CartesianVelocity`	Derivative of the reference trajectory for the CoM. This includes both linear and angular velocities of the base link.
$\ddot{\mathbf{x}}_{imu} \in \mathbb{R}^3$	IMU Linear Acceleration	`LinearAcceleration`	This vector contains the linear acceleration measured by the Inertial Measurement Unit (IMU) sensor of the robot.
$\mathbf{Q}_{imu} \in \mathbb{R}^4$	IMU Angular Position	`AngularPosition`	This vector contains the angular position in quaternion measured by the Inertial Measurement Unit (IMU) sensor of the robot.
${}_{W}\mathbf{p} \in \mathbb{R}^3$	Real ZMP	`ZeroMomentPoint`	Real ZMP with respect to the world.
${}_{W}\dot{\mathbf{p}} \in \mathbb{R}^3$	Real ZMP velocity	`ZeroMomentPointP`	First derivative o real ZMP with respect to the world.
$\in\mathbb{R}\zeta_{W}3$	Real DCM with respect to the world.	`DivergentComponentOfMotion`	Real position of the DCM with respect to the world.
${}_{W}\dot{\boldsymbol{\zeta}} \in \mathbb{R}^3$	Reference DCM Velocity	`DivergentComponentOfMotionP`	Derivative of the real DCM with respect to the world.
${}^{M}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Real CoM Position	`HomogeneousTransformation`	Real position of the CoM. This includes both position and orientation of a virtual link located at the CoM with the same oriantation of the base link.
${}_{W}\dot{\mathbf{X}}_{M} \in \mathbb{R}^6$	Real CoM Velocity	`CartesianVelocity`	Derivative of the real trajectory of the CoM. This includes both linear and angular velocities of the base link.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

2.2 Outputs

Symbol	Name	Type	Description
${}^{\text{CoM}_{\text{cmd}}}_{\text{CoM}_{\text{r}}}\mathbf{T}_{\text{off}} \in \mathbb{R}^{4\times4}$	COM offset	`HomogeneousTransformation`	Offset needed on the COM commanded position to keep balance and track the reference trajectories with the real position.
${}_{\text{W}}\boldsymbol{\zeta}_{\text{d}} \in \mathbb{R}^3$	Desired DCM Position	`DivergentComponentOfMotion`	Adjusted trajectory for the DCM to keep balance.
${}_{\text{W}}\dot{\boldsymbol{\zeta}}_{\text{d}} \in \mathbb{R}^3$	Desired DCM Velocity	`DivergentComponentOfMotionP`	Adjusted velocity for the DCM to keep balance.
${}_{\text{W}}\mathbf{p}_{\text{d}} \in \mathbb{R}^3$	Desired ZMP	`ZeroMomentPoint`	Adjusted trajectory for the ZMP to keep balance and track the desired DCM.

2.3 Inter-Connections

This module receives all the reference trajectories and their derivatives from the CoM trajectory generator module (CoMTG), the IMU sensor data form the Real Robot module (RR), the real ZMP and its derivative from the ZMP module, and all the output from the real CoM module. The output of this module is connected to the Command generator module which combines the offsets with the reference trajectories.

2.4 Common Methods

There are several techniques for biped balance control as shown in [1]. One simple method to stabilize the LIPM model is to define a a proportional ZMP and CoM tracker in the form

$$ {}_{\text{W}}\dot{\mathbf{x}}_{\text{M}_\text{d}} = k_{\text{zmp}}\left({}_{\text{W}}\mathbf{p} - {}_{\text{W}}\mathbf{x}_{\text{M}}\right) + k_{\text{M}}\left({}_{\text{W}}\mathbf{x}_{\text{M}_\text{r}} - {}_{\text{W}}\mathbf{x}_{\text{M}}\right) \tag{2.1} $$

where $k_{\text{zmp}}, k_{\text{M}} \in \mathbb{R}$ are positive gains. Another method is to stabilize the LIPM using LQR-control which results in a control in the form

$$ {}_{\text{W}}\dot{\mathbf{x}}_{\text{M}_\text{d}} = -k_1\,{}_{\text{W}}\mathbf{p} - k_2\,{}_{\text{W}}\mathbf{x}_{\text{M}} - k_3\,{}_{\text{W}}\dot{\mathbf{x}}_{\text{M}_\text{r}} \tag{2.2} $$

where $k_1, k_2, k_3 \in \mathbb{R}$ are optimal gains tunned with LQR.

Finally, one las example is to adjust the desired ZMP position for tracking the reference DCM as described in [2]

$$ {}_{\text{W}}\mathbf{p}_{\text{d}} = k_{\text{d}}\,{}_{\text{W}}\boldsymbol{\zeta}_{\text{r}} - \left(k_{\text{d}} - 1\right){}_{\text{W}}\boldsymbol{\zeta} \tag{2.3} $$

with $k_{\text{d}} < 0 \in \mathbb{R}$. This desired ZMP can the be tracked with a simple PD as

$$ {}_{\text{W}}\dot{\mathbf{x}}_{\text{M}_\text{d}} = k_{\text{zmp}}\left({}_{\text{W}}\mathbf{p} - {}_{\text{W}}\mathbf{p}_{\text{d}}\right) + k_{\text{d}}\,{}_{\text{W}}\dot{\mathbf{p}} \tag{2.4} $$

References

[1] Kajita, Shuuji, et al. Introduction to humanoid robotics. Vol. 101. Springer Berlin Heidelberg, 2014.

[2] Englsberger, Johannes, and Christian Ott. "Integration of vertical com motion and angular momentum in an extended capture point tracking controller for bipedal walking." 2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012). IEEE, 2012..

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Module Description: Command Generator (CMDGENM)

Simon Armleder, Emmanuel Dean, Florian Bergner,
Rogelio Guadarrama-Olvera, and Gordon Cheng

February 14, 2020

1 Module Description

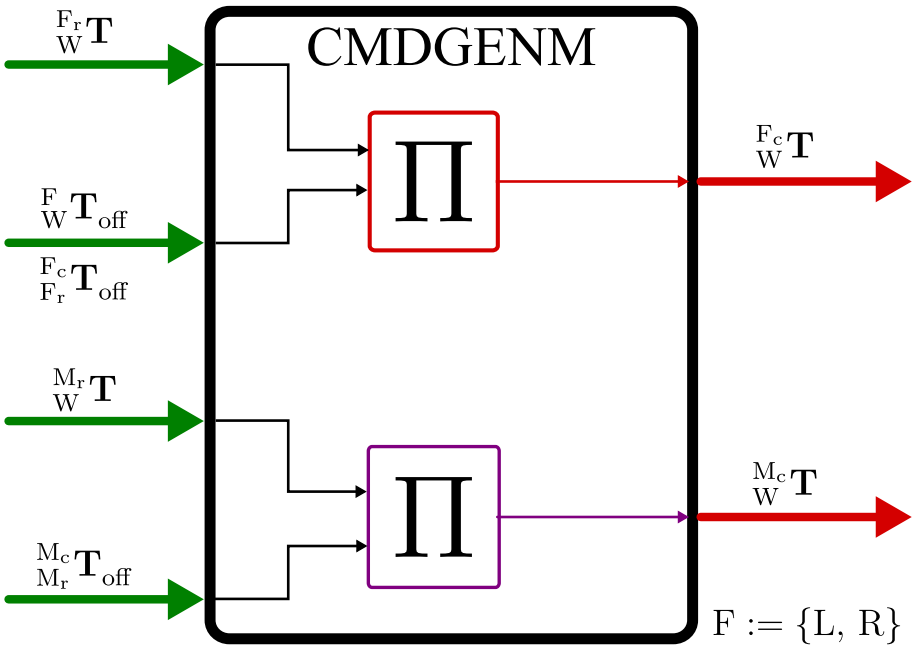

Figure 1.1: Command Generator: This module computes the commanded Cartesian pose for both feet and the center of mass.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

The *Command Generator* module (CmdGen) computes the commanded Cartesian pose for both feet and the center of mass. A command is constructed from a planned reference pose that should be tracked by the robot and an offset that corrects for external disturbances such as e.g. unknown ground conditions. The offset stores relative translational and rotational corrections that are applied to the reference trajectory to ensure tracking.
Finally, the commanded Cartesian poses are send to the *Inverse Kinematics* solver (IK).

2 Module Connections

2.1 Inputs

Symbol	Name	Type	Description
${}^{L_r}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Left Foot Reference Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix contains the reference pose of the left foot ankle coordinate frame L with respect to the world coordinate frame W.
${}^{R_r}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Right Foot Reference Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix contains the reference pose of the right foot ankle coordinate frame R with respect to the world coordinate frame W.
${}^{M_r}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Center of Mass Reference Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix contains the reference pose of the Center of Mass coordinate frame M with respect to the world coordinate frame W.
${}^{L_c}_{L_r}\mathbf{T}_{off} \in \mathbb{R}^{4\times4}$	Left Foot Offset Transformation	`HomogeneousTransformation`	This homogeneous transformation matrix contains the relative offset for the left foot reference Coordinate Frame L.
${}^{R_c}_{R_r}\mathbf{T}_{off} \in \mathbb{R}^{4\times4}$	Right Foot Offset Transformation	`HomogeneousTransformation`	This homogeneous transformation matrix contains the relative offset for the right foot reference Coordinate Frame R.
${}^{M_c}_{M_r}\mathbf{T}_{off} \in \mathbb{R}^{4\times4}$	Right Foot Offset Transformation	`HomogeneousTransformation`	This homogeneous transformation matrix contains the relative offset for the center of mass reference Coordinate Frame M.

2.2 Outputs

Symbol	Name	Type	Description
${}^{L_c}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Left Foot Commanded Transformation	`HomogeneousTransformation`	This homogeneous transformation matrix contains the commanded pose of the left foot ankle coordinate frame L with respect to the world coordinate frame W.
${}^{R_c}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Right Foot Commanded Transformation	`HomogeneousTransformation`	This homogeneous transformation matrix contains the commanded pose of the right foot ankle coordinate frame R with respect to the world coordinate frame W.
${}^{M_c}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Center of Mass Commanded Transformation	`HomogeneousTransformation`	This homogeneous transformation matrix contains the commanded pose of the center of mass coordinate frame M with respect to the world coordinate frame W.

2.3 Inter-Connections

The CmdGen is connected to the output of the *Foot Trajectory Generator* (FTG) which provides a reference pose for both feet. The translational and rotational offsets applied to this reference are received through a connection to the *Foot Compliant Model* (FCM).
The *Center of Mass Trajectory Generator* (CoMTG) provides a reference pose for the center of mass. The CoM offsets are generated by the *Balancer* module.

The commanded output poses for feet and CoM are send to the *Inverse Kinematics* solver (IK) to compute the required joint space coordinates.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

2.4 Common Methods

Both, the reference poses and offsets are represented by Affine Transformation matrices. The pose commands are then obtained by multiplying the references with their corresponding offsets.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

TUM Institute for Cognitive Systems (ICS)

OpenWalker

Module Description: Inverse Kinematics (IKM)

Florian Bergner, Emmanuel Dean, Rogelio Guadarrama-Olvera,
Simon Armleder, and Gordon Cheng

February 14, 2020

1 Module Description

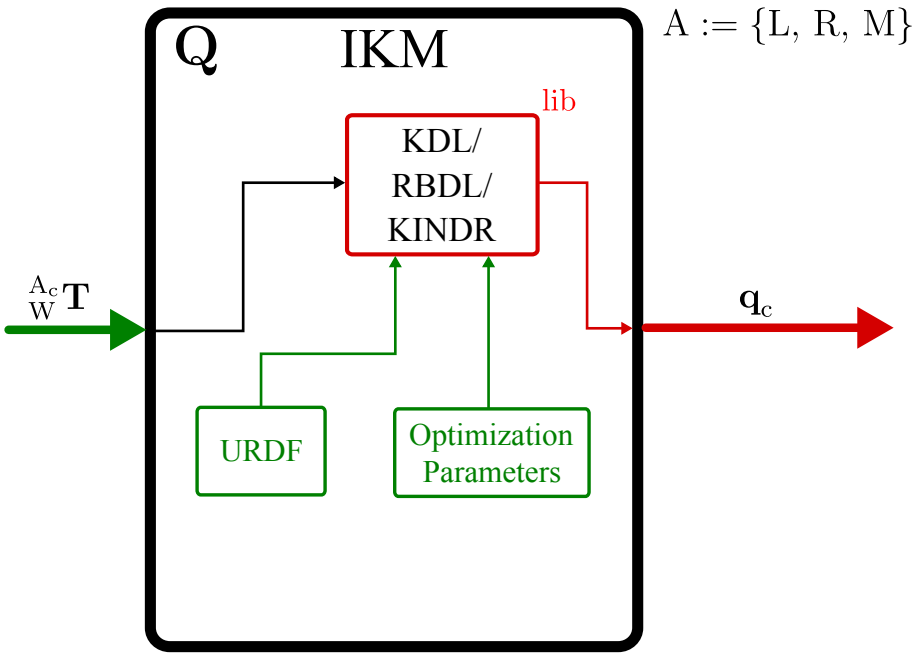

Figure 1.1: Inverse Kinematics module: This module implements the inverse kinematics for the robot.

The *Inverse Kinematics* module (IKM) computes the inverse kinematics of the robot. The inverse kinematics finds a set of joint positions such that a given set of robot end-effectors

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

reach a specified Cartesian position. The OpenWalker framework uses one IKM to compute the commanded joint position $\mathbf{q}_c$ such that the left and right foot, and the center-of-mass (CoM) of the real robot reach the desired Cartesian position ${}^{L_c}_{W}\mathbf{T}$, ${}^{R_c}_{W}\mathbf{T}$, and ${}^{M_c}_{W}\mathbf{T}$.

2 Module Connections

2.1 Inputs

Symbol	Name	Type	Description
${}^{L_c}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Left Commanded Foot Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix transforms coordinates in the left commanded foot coordinate frame L to the world coordinate frame W.
${}^{R_c}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Right Commanded Foot Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix transforms coordinates in the right commanded foot coordinate frame R to the world coordinate frame W.
${}^{M_c}_{W}\mathbf{T} \in \mathbb{R}^{4\times4}$	Commanded CoM Coordinate Frame	`HomogeneousTransformation`	This homogeneous transformation matrix transforms coordinates in the commanded CoM coordinate frame to the world coordinate frame W.

2.2 Outputs

Symbol	Name	Type	Description
$\mathbf{q}_c \in \mathbb{R}^{DOF}$	Commanded Joint Position	`JointPosition`	This vector contains the next commanded joint positions for all the joints of the robot. The OpenWalker framework uses this module input to send position commands to the position controlled real robot.

2.3 Inter-Connections

The inputs of the IKM are connected to the outputs of the Command Generation Module (CmdGenM) which fuses the reference and offset coordinate frames of the left and right foot and the CoM to commanded coordinate frames. The output of the IKM is connected to the Real Robot Module (RRM) which sends the commanded joint positions to the position controlled real robot.

2.4 Common Methods

Similar to the Forward Kinematics Module (FKM), this module uses kinematic parameters such as joint properties (location, type), and link properties (location, length) to build up a rigid multi body system (MBS) that represents the kinematic model of the robot. Using this kinematic model, we can iteratively compute the inverse kinematics using a damped Levenberg-Marquardt method, also known as Damped Least Squares method. Therefore we repeatedly compute

$$\mathbf{q}_{c,\,k} = \mathbf{q}_{c,\,k-1} + \Delta\boldsymbol{\theta} \tag{2.1}$$

$$\Delta\boldsymbol{\theta} = \mathbf{J}^{\top}\left(\mathbf{J}\mathbf{J}^{\top} + \lambda^2\mathbf{I}\right)^{-1}\mathbf{e} \tag{2.2}$$

where

$$\mathbf{J}^{\#\lambda}(\mathbf{q}) = \mathbf{J}^{\top}\left(\mathbf{J}\mathbf{J}^{\top} + \lambda^2\mathbf{I}\right)^{-1} \tag{2.3}$$

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.

is the damped pseudo inverse, $\mathbf{J}(\mathbf{q})$ the Jacobian, and $\mathbf{e}$ the error between the actual Cartesian positions and the target Cartesian positions [1, 2]. The iteration either stops when the error between actual and target Cartesian position is below the tolerance or when the step width

$$||\Delta\boldsymbol{\theta}||_2 \leq \delta_{\text{step}} \tag{2.4}$$

is below the acceptable step tolerance. The parameter λ is the damping factor and has to be chosen carefully.

References

[1] RBDL: An efficient rigid-body dynamics library using recursive algorithms, `https://rbdl.bitbucket.io/de/d92/group__kinematics__group.html#gaa5eabd37ff8b0925d2ecbf49fee1a8a7`.

[2] Di Vito, D., Natale, C., and Antonelli, G. (2017). A comparison of damped least squares algorithms for inverse kinematics of robot manipulators. IFAC-PapersOnLine, 50(1), 6869-6874.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287.