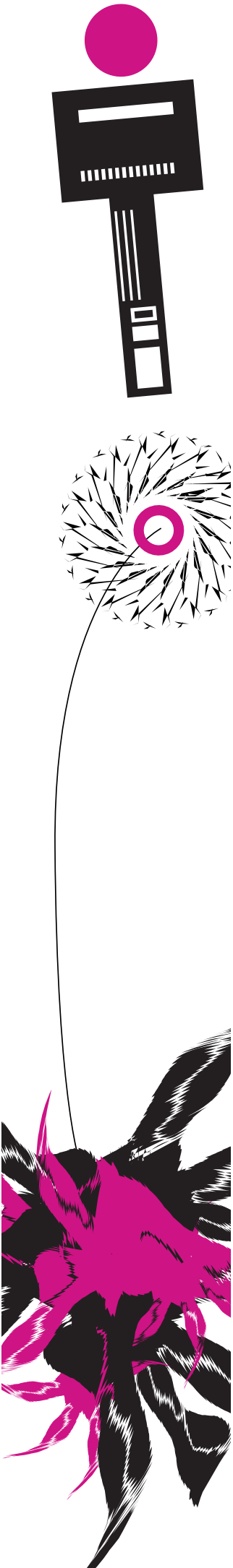


RAM ROBOTICS AND MECHATRONICS

Integrating Energy Awareness into Sequence Control

E.C. (Eddo) Hobert

MSC ASSIGNMENT

Committee:
dr. ir. J.F. Broenink
R. Cobos Mendez, MSc
dr. ing. G. Englebienne

July, 2020

030RaM2020
Robotics and Mechatronics
EEMathCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Summary

Robotic systems require controllers capable of handling not only the complexity of the robotic system itself, but also of its physical interactions with the surrounding environment. The dynamics of physical interactions can be complex and difficult to assess for a control system. Because physical interactions are almost exclusively characterised by energy exchange, integrating energy awareness into robotic control systems benefits their assessment of complex physical interactions.

RaM is working towards fully enabling energy awareness in control systems. The first step towards this objective was taken with the enablement of energy awareness at the loop-control layer. The next step is to integrate energy awareness at the sequence-control layer, which is the aim of the project discussed in this thesis. This thesis presents the architectural design and conceptual implementation of an energy-aware sequence-control layer that is to form the basis for future work on energy awareness at the sequence-control layer and higher control layers.

First, existing sequence-control implementations are analysed to determine connection points for energy awareness. From this analysis, it is concluded that the integration of energy awareness at the sequence-control layer can lead to the improvement of system-level properties, but does require accurate sensor and actuator calibrations, collocation of control signals, and accurate energy-tank budgets. Next, equations involved in calculating interaction energies are analysed, yielding 8 equations containing 4 physical variables: generalised force (effort), generalised momentum, generalised velocity (flow), and generalised displacement. To communicate energies and these 4 variables in a physics-conformal manner, two novel interfaces are proposed: an energy interface and a control interface, respectively.

To demonstrate energy awareness of the sequence-control layer, 5 typical fault scenarios that would each require a different correction approach from the sequence controller are analysed: I. Unexpected Obstacle, II. Sensor or Actuator Failure, III. Incorrect Target Specifications, IV. Incorrect Target Pose, and V. Communication Delay or Loss. Based on the analyses, a list of prioritised design- and implementation requirements is composed. By satisfying these requirements, an energy-aware sequence-control layer or any of its components can be developed and connected to other entities that satisfy the same requirements.

Fulfilling all design requirements, an architectural design of the energy-aware sequence-control layer is presented, consisting of 3 components and 2 interfaces: a setpoint generator, energy estimator, passivity layer, and the above-mentioned energy- and control interfaces. The setpoint generator assesses the control system's energetic state and adjusts the generation of its amplitude-continuous setpoint commands accordingly. The energy estimator provides model-based energy budgets to the passivity layer's energy tank. The passivity layer establishes a passive connection between the sequence-control layer and the loop-control layer. All communications between the 3 components are established by the 2 interfaces.

An energy-aware sequence controller is implemented in simulation to serve as a proof of concept for the architecture design. Fault scenario I, Unexpected Obstacle, is implemented using a virtual wall. Correspondingly, an energy-based collision-detection algorithm has been integrated in the setpoint generator. By conducting and evaluating 4 experiments of different conditional settings, it is shown that energy awareness is successfully integrated in the sequence-control layer and enables collision detection and reaction within 30 ms.

The work presented in this thesis forms an architectural basis for energy-aware sequence control on a road towards more autonomous and safer robotic systems. Future work should focus on expanding energy assessment to cover a broader range of physical interactions and on testing the system outside of simulation—on a physical robot.

Contents

1 Introduction

1.1 Context

Robotic applications that deal with physical interactions require controllers capable of handling not only the complexity of the robotic system itself, but also of its interactions with the surrounding environment. In practise, such controllers are typically applied in situations where the robot's environment is known beforehand, e.g. in a robotic application along a fixed production line. In situations where the environment is unknown, designing a controller capable of handling the dynamics of that environment becomes significantly more difficult. In such a case, robot control is often assisted by human steering, because humans live in the physical world and are adapted to its dynamics. Conversion of steering commands—often in Cartesian space—to control commands in joint space is performed by a *loop controller* that directly controls the robot's actuators. This scenario of human-assisted robot control is illustrated in Figure 1.1.

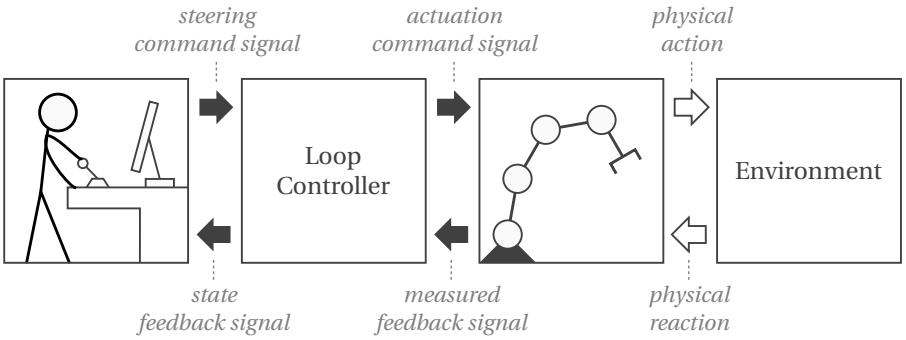


Figure 1.1: A robotic control system based on human steering.

A field in robotics where this situation particularly occurs is (haptic) telemanipulation. In telemanipulation systems, a master sends commands through a communication channel to a slave. The slave follows the master's commands while interacting with some remote environment. We speak of haptic telemanipulation if a human operator is receiving haptic feedback at the master side. Unlike robotic applications along fixed production lines, applications of haptic telemanipulation are often characterised by complex physical dynamics, an unknown environment, and/or high risk manipulations. Fields of application include surgery, nuclear decommissioning, waste management, and manufacturing, like in the work by Rahal et al. (2019) on improving the control autonomy of human-assisted robotic cutting systems.

Control systems are increasingly becoming more autonomous. This development can be prominently observed through the advances being made in autonomous vehicles, which are transitioning away from being human-controlled systems. Since there is also still a dependency on human steering in robotics, new paradigms are needed to improve the autonomy of robotic control systems. To this end, Brodskiy (2014) took an energy-based approach. A key concept behind the use of energy in control is that physical interactions are almost exclusively characterised by energy exchange (Folkertsma and Stramigioli, 2017), allowing an energy-aware controller to assess physical interactions. *Energy awareness* means to plan and perform actions taking into account a system's energetic state. The research by Brodskiy (2014) shows that enabling energy awareness in robotic control is a means to improve *system-level properties*, such as autonomy, but also performance, fault tolerance, and safety.

Control systems can be divided into multiple control layers, with each layer serving a different purpose, such as the left two blocks in Figure 1.1. A generally-applicable diagram rep-

resentation of layered control was presented by Bezemer (2013) and is shown in Figure 1.2. The diagram shows the possible layers in a control system and an indicative distribution of timing guarantees. The Loop Controller in Figure 1.1 corresponds to the Loop Control layer in Figure 1.2. The human operator in Figure 1.1 fulfils the functionality of higher control layers in Figure 1.2, i.e. 'Sequence Control', 'Supervisory Control & Interaction', and 'User Interface'.

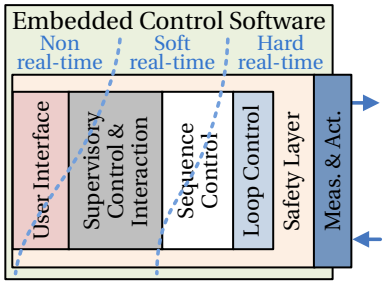


Figure 1.2: A general software-architecture representation for layered embedded control systems (Bezemer, 2013).

1.2 Problem Statement

Since energy awareness can improve system-level properties, it can be beneficial to enable energy awareness for the full control stack shown in Figure 1.2. The University of Twente's Robotics and Mechatronics (RaM) research group[1] has been working on integrating energy awareness into the loop-control layer. Continuing along this path, the next steps towards full energy awareness in robotic control are about integrating energy awareness into higher control layers, starting with the *sequence-control layer*.

1.3 Project Goals

The main goal of this project is to take the next step in energy-aware robotics; namely,

enabling energy awareness at the sequence-control layer.

This goal has been divided into the following subgoals:

- Identifying how energy awareness can be enabled and harnessed at the sequence-control layer.
- Defining design and implementation requirements for energy-aware sequence control.
- Designing and implementing the architecture of an energy-aware sequence-control layer.
- Demonstrating improved system-level properties through energy awareness.

By achieving these goals, the project forms an architectural basis for energy awareness at the sequence-control layer. Figure 1.3 illustrates the addition of an energy-aware sequence controller to the example of human-assisted robotic control. Herein, on top of receiving energetic feedback, the human operator is enabled to focus more on higher-level control operations, making the control system better equipped to handle physical interactions of higher complexity. In essence, the aspired contribution of this project is to provide a generic solution on improving properties (such as autonomy, stability and safety) of any control system that deals with physical interactions, including—but not limited to—human-assisted robotic control systems.

[1] RaM website: *https://www.ram.eemcs.utwente.nl/*

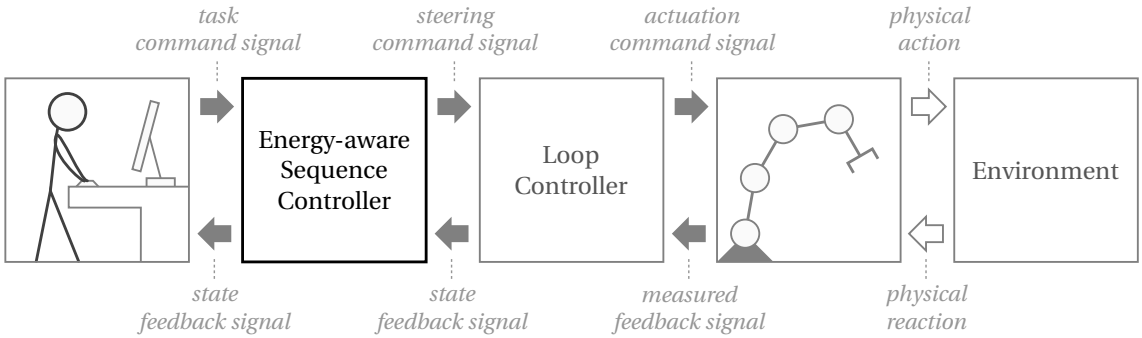


Figure 1.3: A robotic control system in which an energy-aware sequence controller steers the loop controller, thereby allowing a human operator to put more focus on higher-level tasks (compared to Figure 1.1).

1.4 Approach

The approach for reaching the project goals starts by exploring available information. Topics hereof are: good design practises for robotic control systems, the concept of layered control, energy-based concepts of physics, and previous work on the just-mentioned topics in relation to sequence control. This exploration of background information is covered by Chapter 2.

Following up on the background exploration, a list of sequence control example projects is analysed for common structures, possible connection points to energy awareness, and system properties to be improved. These analyses of other projects are concluded by a list of pros and cons on integrating energy awareness into the sequence-control layer. Because this thesis focusses on control systems that deal with physical interactions, an analysis is performed to define a physics-conformal control interface. Furthermore, for the communication of energy data, an analysis to define an energy interface is performed. As modern controllers are digital, discrete-time equations (compliant to the control- and energy interfaces) are analysed for the computation of energy exchanges. As a final part of the analyses, use cases for the implementation of energy-aware sequence control are analysed. The project analyses are presented in Chapter 3 and concluded by a list of prioritised requirements for designing and implementing an energy-aware sequence-control layer.

Conform to the design and implementation requirements, the architecture of an energy-aware sequence-control layer is designed, followed by a conceptual implementation serving as a proof of concept of that design. The design is physics-conformal and good practises in design are applied. The design and implementation are treated in Chapter 4. Using the design's conceptual implementation, experiments are performed to evaluate energy awareness of the implemented sequence-control layer and its effects on system-level properties, as presented in Chapter 5. Finally, Chapter 6 concludes the project on the basis of the project goals and lists recommendations for future work.

2 Background

This chapter presents background information on the project's goal of enabling energy awareness at the sequence-control layer. First, good practises in design of robotic control software are treated. Following the good practises in design, the basics behind the different types of control layers and how they cooperate are detailed. Then, concepts related to energy exchange during physical interactions are explained together with possible connections to the control layers. Finally, these topics come together in a section about related work on energy awareness in the layered control stack.

2.1 Good Practises in Design

The list of potential applications for robotics seems endless. Robotics is often applied with custom hardware and/or software. Since custom solutions typically come at higher time, monetary, and repair costs than commercial off-the-shelf (COTS) solutions, it makes sense to look for better approaches on working through the seemingly endless list of potential applications. Good practises in design are needed for a more structured and future-proof approach. One possible approach is to apply Component-Based Software Development (CBSD), as was done in the robotics works by Bezemer (2013); Brodskiy (2014); Tadele (2014). In CBSD, systems are composed from reusable components comprising a trade-off between COTS and custom development (Brugali and Scandurra, 2009). Reusable components in CBSD have the following five essential characteristics (Sommerville, 2016):

Composable components communicate data through publicly available interfaces.

Deployable components are self-contained and can operate individually.

Documented components are fully explained in written text.

Independent components are composable and deployable without external dependencies.

Standardised components conform to a standard model or architecture.

CBSD facilitates an efficient workflow, because it prevents engineers from having to spend time on developing software components that may already exist (reinventing the wheel). An additional benefit is that when a component is being reused, the amount of engineers with knowledge about that component grows, which adds to the accessibility and maintainability of that component. Ideally, a CBSD component should be compliant to all possible systems. However, naturally, there is a limit to the extend of compliance of components due to differences in interfaces, e.g. specialistic systems using a proprietary interface or existing systems using an outdated configuration. Therefore, CBSD is about finding the best trade-off between being too specific (less reusable) and too generic (less valuable):

> *"The highest-quality component will never be reused if the function it offers is useless. The most needed component will never be reused if it is unreliable, slow, hard to understand. The highest-quality and most needed component will never be reused if the interface is not compatible."*
>
> — Brugali and Scandurra (2009)

An important topic in CBSD is separation of concerns, in which a system is separated on the basis of specific functional distinctions (concerns). Applying separation of concerns in the development of control software adds to the reusability of a system and allows work on the system to be focussed at one specific system functionality at a time. A set of five concerns (5Cs) for the robotics domain was presented by Bruyninckx et al. (2013), as shown in Figure 2.1.

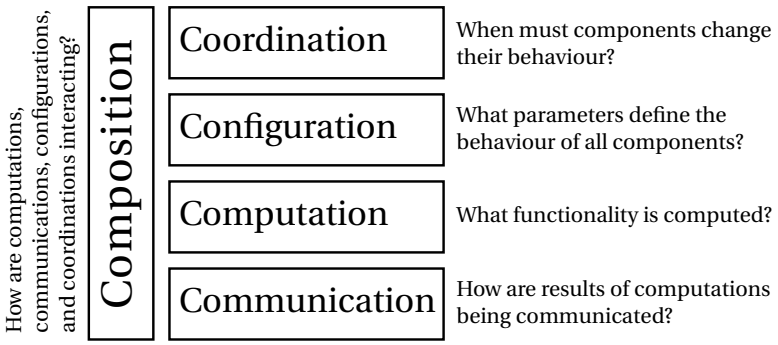


Figure 2.1: Separation of five concerns (5Cs) (Bruyninckx et al., 2013).

2.2 Control in Layers

The concept of separating a control system in layers was introduced in Section 1.1 alongside Figure 1.2. Because of its modular nature, layered control can be applied in conformity with the good practises in design described in Section 2.1.

Layered control systems, or Distributed Control Systems (DCSs), comprise a hierarchy of controllers that split the control system into several sections of responsibility (Greeff and Ranjan, 2004; Bolton, 2015). DCSs are integrated systems based on the concept of decentralisation and ensure a high degree of reliability, which is why they are typically used where significant amounts of control and a high degree of fault tolerance and redundancy are required (Greeff and Ranjan, 2004).

The general architecture representation for layered control systems in Figure 1.2 only considers the software side of a control system. A complete control system also includes *Input/Output Hardware* and a system to be controlled, called the *Plant*. This is shown in Figure 2.2, as presented by Bezemer (2013)[1]. Note that the Embedded Control Software section in Figure 2.2 is a copy of Figure 1.2. Even though this thesis focusses on the sequence-control layer, during development of any control layer in software, it is important to keep in mind the full control system including I/O Hardware and the Plant.

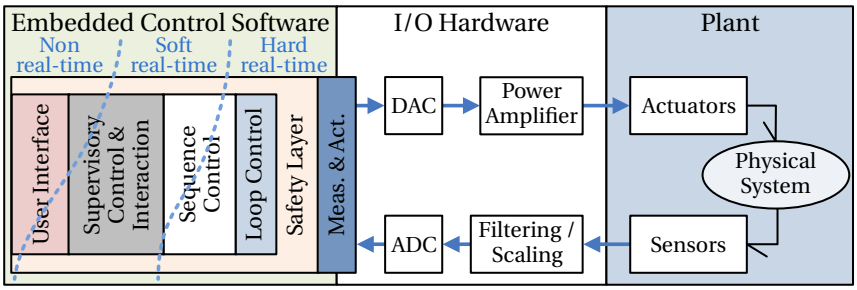


Figure 2.2: A complete general system-architecture representation for layered embedded control systems (Bezemer, 2013).

Control systems have timing requirements depending on execution criticality. These can be either non-, soft-, or hard real-time requirements. At lower control layers, e.g. loop control, precise time intervals are critical for reliable performance, in which case hard real-time is required. If some slack in timing precision, for instance at higher control layers, does not affect reliability and safety, then soft real-time or even non real-time can be sufficient.

[1] In fact, this figure has been used within the RaM group for years and was originally inspired by Bennett (1988). An early version can be found in the work by Groothuis et al. (2009), after which an upgraded version was presented by Broenink et al. (2010). Based on that, the current version in Figure 2.2 by Bezemer (2013) contains an update on how the real-time guarantees are distributed over the different control layers.

Each of the control layers presented in Figure 2.2 performs control actions under different considerations of the controlled system. The higher the control layer, the the higher the abstraction from the continuous-time physical world. Below follows a description of each of the embedded control software layers and how they cooperate.

Safety Layer oversees all control layers and checks their inputs and outputs for unwanted signals. Its purpose is to prevent the propagation of unintended control component behaviours.

User Interface is the highest control layer. It communicates information to the user, takes and processes inputs from the user, and passes the user's commands on to the supervisory control & interaction layer.

Supervisory Control & Interaction deals with complex algorithms such as path planning, image processing, or environment mapping. Tasks are calculated and commanded to the sequence-control layer for execution. State feedback, containing information about the control system, is returned to the user interface as feedback for the user. Interaction takes place through the processing of sensor data or user inputs, but also through communication with other (cyber-physical) systems.

Sequence Control is the main topic of this thesis. Like the User Interface and Supervisory Control & Interaction layers, sequence control is considered a form of higher-level control. The sequence-control layer is often implemented as a finite state machine and typically deals with longer running tasks than loop control, e.g. trajectory planning. Tasks are commanded by the supervisory-control layer. Based on these tasks, the sequence-control layer calculates setpoints for loop control to follow. Besides setpoints, commands can also be sent to loop control in the form of changes to control parameters. State feedback is returned to the supervisory control & interaction layer and comprises control information of the sequence-control layer, but possibly also of the loop-control layer and/or the plant. In turn, sequence control receives feedback from loop control and may also receive sensor data (e.g. from cameras, lasers, IMUs, GPSes, etc.). Sequence control components can require either soft- or hard real-time. If missed deadlines do not jeopardise task execution, soft real-time is sufficient. If they may lead to critical failures, hard real-time is required.

Loop Control is responsible for controlling the plant (a continuous-time dynamic system). The loop-control layer receives setpoint commands from the sequence-control layer and measured sensor feedback from the measuring & actuation layer. It calculates actuation commands based on these setpoints and measurements. The actuation commands are sent to the plant's actuators (via the measuring & actuation layer). State feedback is sent to the sequence-control layer and can be in the form of actuation commands and/or state information obtained from the plant.

Measuring & Actuation provides an interface between the control software and the system hardware by filtering and scaling incoming and outgoing signals before passing them through. Outside of the embedded control software, it is connected to digital-to-analogue (D/A) and analogue-to-digital (A/D) converters, connecting the discrete-time control stack to the continuous-time plant.

Following the descriptions listed above, an illustration of the propagation of command and feedback signals from the user-interface layer to the measuring & actuation layer is shown in Figure 2.3.

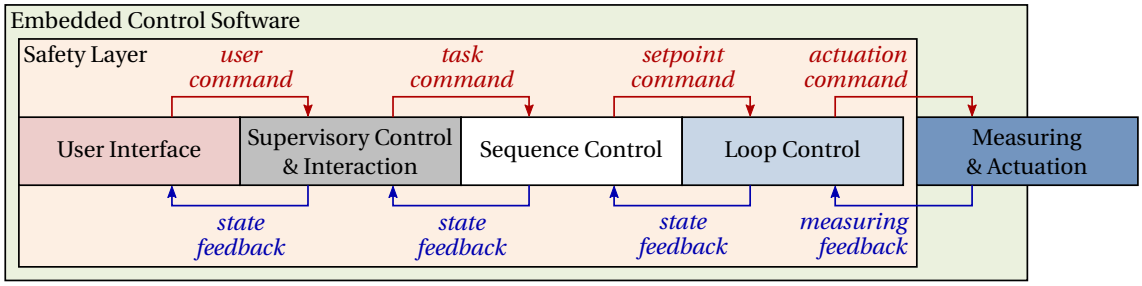


Figure 2.3: Command and feedback signals between adjacent control layers in the control stack of Figure 2.2.

2.3 Principles of Energy Exchange

This section treats how the observation stated in Section 1.1, that "interactions are almost exclusively characterised by energy exchange" (Folkertsma and Stramigioli, 2017), relates to the development of an energy-aware sequence-control layer. Relevant background information comprises the quantities involved with physical interactions, basic energy calculations, and power ports. Because this is important—yet basic—information, it is presented in Appendix A.

2.4 Related Work

Previous work has been done on robotic control systems that follow good practises in design, control in layers, and energy exchange principles presented in Sections 2.1 to 2.3. Such work, related to the project goal of enabling energy awareness at the sequence-control layer, is described here.

Franken (2011) worked on port- and energy-based control of haptic telemanipulation systems. Control stability was achieved through a passivity layer. Passivity layers function like a safety layer that enforces system passivity—an energy-based measure of stability. Passivity was first defined by Willems (1972) and dictates that the energy in a system is always less than or equal to the initial energy in that system. Hence, a passive system is either dissipative or lossless, such as mass-spring-damper systems. An important property of passivity is that any power-conserving interconnection of passive subsystems forms a new system that is again passive (Willems, 1972).

In fact, robot passivity is a necessary condition for system stability during interaction between a robot and an unknown environment, but does not guarantee stability if that environment is active (Folkertsma and Stramigioli, 2017). Activity is the opposite of passivity, i.e. a system of increasing energy[2]. In case of interaction with an active environment, sufficient damping should be added to achieve system passivity (Groothuis et al., 2018). Brodskiy (2014) applied a passivity layer in the context of the control stack of Figure 2.2 and evaluated the potential benefits of energy-awareness. He concluded that energy-based concepts offer an effective approach to analyse system performance and that system performance can be greatly enhanced if a control strategy incorporates information about energy states.

In order to provide the passivity layer's energy tank with sufficient energy, Brodskiy (2014) used an energy estimation component to determine energy budgets sufficient for task execution. According to Groothuis et al. (2018), "energy budget allocations for generic tasks and movements can only usefully be done for finite time windows... It makes sense to consider the energy requirements during each time step and providing an energy budget suitable for that time step".

[2] Universally speaking, any physical system is passive by the first law of thermodynamics, but may seem active because we restrict ourselves to certain time- and spatial bounds (Stramigioli, 2001). For example, a battery-powered system can appear active as the battery performs like a constant voltage source, but this is a time-bounded state since batteries store only a finite amount of energy.

Accordingly, the energy estimation block by Brodskiy (2014) determined energy budgets per time step by running a simultaneous simulation of the controlled system dynamics. He found that incorrect estimation of the energy budgets results in a negative impact on system performance by the passivity layer.

Passivity layers have also been applied by RaM in a contribution to RobMoSys. RobMoSys[3] is an effort on establishing an open, multi-domain, modular and sustainable software ecosystem for the European robotics community with the aim of improving accessibility and efficiency in robotics development. RaM has been contributing to RobMoSys by working on integrating energy awareness into the loop-control layer to guarantee passivity and enhance safety for the plant and its surroundings. Work has been done on designing and implementing energy guards, which function as local passivity layers, to encompass each loop-control component.

The next aim of RaM is to extend energy awareness from the loop-control level to the system level by integrating it into the complete motion control stack. The purpose thereof being to 1) provide information on fault detection and fault handling to the supervisory and sequence control levels, and 2) to determine system-wide energetic properties for ensuring that not only local, but also global passivity and stability are achieved. The plan is to communicate energy data of the loop-control energy guards via an all-overseeing 'energy supervisor' layer to the sequence-control layer. The energy supervisor should ultimately function like an energy-based safety layer in the control stack; communicating and re-assigning energies to and from each control layer. These developments are in line with the goals of the project treated in this thesis. Though development of an energy supervisor is not in the scope of this project, it may later be connected to the work produced in this project.

[3] RobMoSys web page on passive control: *https://robmosys.eu/wiki/community:intrinsically-passive-control:start*

3 Analysis

This chapter focusses on the project's subgoal of "defining design and implementation requirements for energy-aware sequence control", stated in Section 1.3. A list of six projects that make use of sequence control is analysed for common sequence control structures and connection points to energy awareness. The results thereof are used to analyse interface requirements and propose two interfaces: a control interface, and an energy interface. Following the two interfaces is an analysis on energy exchange computations. Furthermore, possible use cases for evaluating energy awareness of the sequence-control layer are analysed to identify implementation requirements. This chapter is concluded by a list of design requirements and a list of implementation requirements.

3.1 Connection Points for Energy Awareness in Sequence Control

In order to determine requirements for designing and implementing an energy-aware sequence-control layer, additional information is needed to verify how sequence controllers are applied in practise. Therefore, six projects that made use of sequence control are analysed below. Their control signals are identified for comparison to the definitions of layered control in Section 2.2. Furthermore, because Sections 1.1 and 2.4 suggest that implementing energy awareness can result into an improvement of system-level properties, system-level properties of each project are evaluated to determine how energy awareness could have been beneficial in those projects. The results of these evaluations can be used as a comparison to the system-level properties of the energy-aware sequence-control layer developed in this work.

System-level properties are properties at the system level, i.e. of the full control system (shown in Figure 2.2) consisting of embedded control software, input/output hardware, and the plant. Properties mentioned in Sections 1.1 and 2.4 are passivity, stability, fault tolerance, safety, performance, and autonomy. System-level properties—rather than sequence-control level properties—are considered, because it is not just the sequence-control layer, but the full system that should have satisfactory properties. Consider, for example, the property of performance. One could develop a sequence controller that performs with a satisfactory speed. However, if this speed is too fast for the loop controller or plant to keep up, the system as a whole will not perform satisfactorily. Therefore, instead of locally, it is ultimately more important to consider properties globally, i.e. at the system level.

Other more relevant system properties can be thought of, e.g. robustness, resilience, versatility, predictability, etc. Alami et al. (2006) narrowed their evaluation of developments in the field of physical human-robot interaction (pHRI)—a field related to the project discussed in this thesis—down to three main properties:

- Safety
- Dependability (availability, reliability, integrity, maintainability)
- Performance (speed, accuracy)

To analyse projects on an equal basis, these three properties also are used for the analyses of the six projects below. The analyses per project are structured as follows:

#. **[Project name].** [Purpose of the system].
   (a) **Measuring.** [The sensors / measurement signals that were used.]
   (b) **Actuation.** [The actuators / control-command signals that were used.]
   (c) **Notes.** [Relevant notes specific to the project, e.g. notable design choices or problems that occurred.]

(d) **Improving System-Level Properties by Implementing Energy Awareness.**

- **Safety.** [Evaluation of the system's safety property and how energy awareness could have been beneficial. Note that, since passivity is an energy-based measure of stability, a passive system is a stable system. Stability, in turn, contributes to safety. Also note that safety can be regarded a subproperty of dependability, but is considered separately here.]
- **Dependability.** [Evaluation of the system's dependability property and how energy awareness could have been beneficial. Note that this considers subproperties availability, reliability, integrity, and maintainability, if applicable. Also note that dependability overlaps the robustness property, which entails availability, reliability, and maintainability as well (Alami et al., 2006).]
- **Performance.** [Evaluation of the system's performance property and how energy awareness could have been beneficial. Note that this considers speed and accuracy, but also versatility, for example.]

To be able to extract useful information about the potential consequences of integrating energy awareness into sequence control, the main criteria for the selected projects is that they make use of sequence controllers that are not already energy-aware. Furthermore, to extract information that is widely applicable, the control systems of the projects have been selected to differ in size and shape, number of joints, type of actuation, and/or type of physical-interaction application.

Correspondingly, the following projects (in reverse chronological order) have been selected: 1) a human-inspired robot hand (Delgado et al., 2017), 2) a pipe inspection robot; PIRATE (Morales, 2016), 3) a soft landing pneumatic drive (Pfeffer et al., 2016), 4) an autonomous ATV (Bardaro et al., 2014), 5) a humanoid robot; TUlip (Lootsma, 2008), and 6) an exoskeleton (Kawamoto and Sankai, 2004). No more than these six projects have been selected, because they are expected to yield sufficient information. The analyses of the six projects are listed below:

1. **Robot Hand.** Delgado et al. (2017) extracted object deformation properties with a human-inspired robot hand[1] for tactile measurements and used those determined properties for manipulating the object. A task planner (sequence controller) was used to control positions and forces.

   (a) **Measuring.** 18 resistive force/pressure sensors distributed over the palm and fingers of the robot hand. Position data of the joints is available. A Kinect is used to localise the object that is to be grasped.

   (b) **Actuation.** 5 articulated fingers and 20 Degrees of Freedom (DOFs). All joints can be moved by the 20 motors in the hand base[2].

   (c) **Notes.** The research aimed "to imitate the behavior of human beings, in which the applied forces by the fingers are changed when the human estimates the rigidity of a body and when the fingers react to unexpected movements of the object to keep the contact points" (Delgado et al., 2017). Force and displacement data are used separately in the control algorithm.

   (d) **Improving System-Level Properties by Integrating Energy Awareness.**

   - **Safety.** By not keeping track of energy exchange, passivity can be broken. This could result in unstable and unsafe behaviour, especially since the system interacts with unknown objects. An energy-aware sequence controller can en-

[1] Shadow Dexterous Hand web page: *http://www.shadowrobot.com/products/dexterous-hand/*

[2] Technical specification: *https://www.shadowrobot.com/wp-content/uploads/shadow_dexterous_hand_technical_specification_E_20190221.pdf*.

force passivity and thereby stability, leading to a safer handling of unknown objects.

- **Dependability.** The authors mention that support from a vision system would be needed to reliably recognise more complex deformable objects such as cloths. Reliability against these unknown objects may be improved through energy-awareness. Energy-aware control would merge the force and displacement data (which were used separately) to determine energy exchange. This may make it possible to increase controller sensitivity, which would also be an improvement in performance.
- **Performance.** Furthermore, vision systems perform relatively slow. An energy-aware control approach is faster and would lead to higher responsiveness. Additionally, given that interactions are almost exclusively characterised by energy exchange (Folkertsma and Stramigioli, 2017), an energy-aware sequence controller could improve the assessment of physical interactions. This can be used to improve the system's performance on its aim of imitating human behaviour.

2. **PIRATE.** Morales (2016) used sequence control to control a pipe inspection robot (PIRATE) consisting of 8 links and 7 joints performing tasks such as clamping, moving around a corner, and driving inside pipes with a diameter slightly larger than the robot itself.
   (a) **Measuring.** Sequence control receives feedback from joint encoders (angles and efforts) and an IMU sensor.
   (b) **Actuation.** The systems contains 2 motors for clamping, 2 motors for camera pan and tilt, 1 motor for rotation, and 6 drive motors (1 for each wheel).
   (c) **Notes.** The control mode can be either torque, velocity, or position. Even though both torque and angular-position feedback are available, only the angular position is used because of unreliable torque measurements. Regardless, a combination of torque commands and angular-position feedback could have been used to estimate energy exchange.
   (d) **Improving System-Level Properties by Integrating Energy Awareness.**
      - **Safety.** Communication delays may occur between the robot in the pipe and the remote human operator. Communication delays can lead to instability, which may harm the robot (Brodskiy, 2014). Energy-aware control can enforce passivity (and therefore stability) and make the system more safe by preventing such harms from taking place. This would also improve robustness.
      - **Dependability.** If torque control would be used, the commanded torques could be compared to the measured torques to filter out incorrect measurements, but the commanded torques could also directly be used to calculate energy exchanges. With energy exchange data, incorrect measurements could also be detected and filtered out. Detecting and filtering out incorrect data attributes to dependability in the form of reliability.
      - **Performance.** The clamping torque is estimated based on a pre-defined fixed pipe diameter. This is a static approach that does not allow for pipe dimension variations. By keeping a fixed potential energy, an energy-aware sequence controller could adapt to any pipe dimension that fits the robot without requiring pre-knowledge, improving its performance by making the system more versatile.

3. **Pneumatic Drive.** Pfeffer et al. (2016) used a linear pneumatic drive with combined position feed-forward and pressure feedback in sequence control to achieve soft landing.

(a) **Measuring.** Pressure sensors and low quality short range position sensors. The position sensors have a limited range and can only measure near the start and end of the trajectory, resulting in a measurement gap.

(b) **Actuation.** One linear pneumatic drive.

(c) **Notes.** The aim is to achieve soft landing despite the limitation on available position sensor data. This was achieved by using a control strategy that combines position feed-forward and pressure feedback. A measurement gap is similar to a temporary communication loss in the sense that they both result in a data gap over a certain time interval.

(d) **Improving System-Level Properties by Integrating Energy Awareness.**

- **Safety & Dependability.** According to Brodskiy (2014), energy-aware control is robust against time delays and can ensure passivity also during a communication loss. This may be extended to robustness against a data gaps as well. Robustness relates to reliability (and, hence, dependability) and passivity is a measure of stability, affecting safety.
- **Performance.** Energy-budget estimations, such as in Brodskiy (2014), could be used to predict the required amount of energy for the task to achieve soft landing. This may improve the computation speed and the accuracy of the system.

4. **Autonomous ATV.** Bardaro et al. (2014) developed a control architecture comprising a planner (supervisory control), trajectory follower (sequence control) and low-level control system (loop control) to create an autonomous All-Terrain Vehicle (ATV).

(a) **Measuring.** GPS, IMU, Magnetometer, Odometer.

(b) **Actuation.** 3 servo motors used for steering, throttle and braking.

(c) **Notes.** System stability was compromised due to the Robot Operating System (ROS) software introducing delays in the trajectory control loop. A pose predictor was introduced by the authors to resolve the issue.

(d) **Improving System-Level Properties by Integrating Energy Awareness.**

- **Safety & Dependability.** Energy awareness could be used to ensure passivity and, hence, stability and safety. This can improve robustness (and therefore dependability) against the communication delays that were experienced, as indicated by Brodskiy (2014).
- **Performance.** An energy-aware controller could implement minimum energy control to extend the system's longevity on a single battery charge.

5. **TUlip.** Lootsma (2008) worked on a humanoid robot (TUlip) that stands up—i.e. moves from a prone position to an upright position—using a state machine sequence controller that changes the setpoints and parameters of 12 PID controllers.

(a) **Measuring.** The sequence controller receives joint-position feedback. The loop controller receives joint position- and velocity feedback.

(b) **Actuation.** There are 12 motorised joints. The PID loop controllers calculate the motor-torque commands as a function of the measured joint positions and velocities.

(c) **Notes.** There was an issue in which "the C-code generated from the model of the body generates errors as soon as actuation of the joints becomes too large. These errors occur because parts of the model are based on the timing of zero-crossings" (Lootsma, 2008). Several fixes were attempted, but all failed. Commanded torque and measured (angular) velocity data could be used to determine energy exchange and enable energy awareness.

(d) **Improving System-Level Properties by Integrating Energy Awareness.**

- **Safety & Dependability.** An energy-aware sequence controller can guarantee passivity, limit the total amount of energy, and distribute the available energy over the actuators. Errors that originate from assigning excessive energies to the actuators could thereby be prevented, which would improve both reliability and safety.
- **Performance.** Energy-aware control could allow the system to assess physical interactions. This can be beneficial in adding functionalities to the system besides standing up, which would improve versatility of the system.

6. **Exoskeleton.** Kawamoto and Sankai (2004) implemented a finite state machine for the sequence-control level of an exoskeleton meant to physically assist a human during walking.

(a) **Measuring.** Rotary encoders at the hip and knee joints, pressure sensors for force measurements at the front and back of the sole of each shoe, and an EMG sensor on both upper legs.

(b) **Actuation.** Torque-controlled actuators at the hip and knee joints. The loop controller calculates torque as a function of the hip or knee angular position, velocity, and/or acceleration (differing per actuator) obtained by the rotary encoders.

(c) **Notes.** The state-machine sequence controller only uses feedback from the force sensors in the shoes and the system only functions when the human is walking, i.e. not when standing still or performing movements other than walking. Commanded effort and measured flow or generalised-displacement data is available at the joints and could have been used to determine energy exchange and enable energy awareness.

(d) **Improving System-Level Properties by Integrating Energy Awareness.**

- **Safety.** Safety is paramount under human-robot interaction. Since system stability influences safety and since passivity is a necessary condition for robot stability during interaction with an unknown environment (Folkertsma and Stramigioli, 2017), the exoskeleton system should function passively. This requires energy-aware control. Additionally, energy awareness could allow for the setting of power- and energy limits to ensure safe human-robot interaction, like in the work by Tadele (2014).
- **Dependability & Performance.** Actuator torques that are a function of the joint angular velocity and acceleration are equal to zero in situations without movement, even when this is not desired, e.g. when keeping a foot still in the air. This could be solved with a more complex control algorithm, which would improve performance at the cost of maintainability and possibly integrity. Instead, implementing energy awareness could be a more dependable and versatile solution. An energy-aware controller can assess both static and dynamic situations through potential- and kinetic energies, respectively, and command adequate assistance. Though, this also could go at the cost of requiring more complex control (lower maintainability and possibly integrity) as well, possibly requiring additional computational resources.

In line with Section 2.2, sequence controllers in the examples above typically send setpoints to a loop controller. This is based on some form of trajectory planning or task execution with feedback coming from various types of possible sensors. In turn, the loop controller typically outputs a generalised force—i.e. effort—command based on a generalised displacement (or sometimes generalised velocity—i.e flow) input feedback.

The command and feedback data can be used to calculate energy exchange (Section 2.3) for the enablement of energy-aware control, but that does require accurate sensor and actuator calibrations. The amount of mechanical energy in a robot is always less than the energy outputted by a controller, because some of the output energy gets converted into thermal energy through friction. This is passive and, thus, safe behaviour. However, passivity cannot be guaranteed if sensors or actuators are not accurately calibrated, because the amount of energy exchange calculated by the controller is then also inaccurate. This can lead to active behaviour with a controller injecting energy where none is needed. For the same reasons, collocation of the actuation commands and measurement feedback data—typically achieved through collocation of actuators and sensors—is also required, as mentioned in Section 2.3. Even more, inaccurate calibrations and non-collocation result in unreliable system performance.

Passive system behaviour cannot be guaranteed with collocations and accurate calibrations only, though. A proven method of enforcing system passivity is through the use of passivity layers, described in Section 2.4. Reusing proven concepts, like passivity layers, is a good practise in design (Section 2.1) and can therefore also be applied in this project. When using passivity layers, caution should be taken with regards to energy-budget estimations, because incorrect estimations negatively impact system performance (Brodskiy, 2014).

Summarising, the integration of energy awareness at the sequence control layer can be characterised by the following pros and cons:

Pros

- Allows physics-conformal control.
- Improved assessment of physical interactions.
- Improvement of system-level properties, for example and among others:
  - Passivity
  - Stability
  - Safety
  - Dependability
  - Robustness
  - Autonomy
  - Performance

Cons

- The control algorithm may become more complex.
  - Additional computational resources may be needed.
- [effort or generalised momentum] and [flow or generalised displacement] data must be available to calculate energy exchange.
- Accurate sensor and actuator calibrations—alongside collocation (Section 2.3)—are required to reliably calculate energy exchange and guarantee passivity.
- System performance depends on adequate energy budgeting for the passivity layer.

3.2 Energy-based Interfaces

Following good practises in design (Section 2.1), data communication within the energy-aware sequence-control layer should be generic and standardised to allow for composable design. To achieve this, two interfaces are explored in this section: a control interface and an energy interface.

3.2.1 Control Interface

Following Section A.2, Section C.1 briefly explains a control system consisting of a sequence-control layer, loop-control layer and plant on the basis of bond-graph principles. Bond-graph principles are relevant for the loop-control layer, because its energy-exchange concern is focussed on closely resembling real-world physics due to its close connection to the physical

plant. An energy-aware sequence-control layer is also concerned with energy exchange, but primarily for using its data to adjust the control strategy. Hence, both control layers are concerned with energy exchange, but for different purposes. This is in line with the notion in Section 2.2 that higher control layers handle higher abstractions from the physical world.

Ultimately, bond-graph entities, such as power ports and power bonds, should be excluded from sequence-control design because of the following three reasons:

1. Sequence control receives its tasks from the supervisory-control layer. These tasks may not be power-continuous and can be more abstract from the physical world than the quantities listed in Table A.1. Even though a task may, for example, be a generalised-displacement setpoint, it could also discretely be "move forward". Such a task cannot be sent through a power bond.
2. Similarly, sequence control may receive sensor data that could, for example, come from cameras, lasers, IMUs, GPSes, etc. As a result, the sequence-control layer does not strictly represent an impedance or admittance within the control system, because it can receive multiple inputs that are not necessarily related to physical energy exchange. It can still resemble impedance or admittance functionality to loop control, though, by communicating the appropriate power-conjugated effort-flow pair.
3. Generalised momenta ($\boldsymbol{p}$) and generalised displacements ($\boldsymbol{q}$) can also be used to determine energy exchange. However, they are not compatible to bond-graph notation, which only considers generalised forces (efforts, $\boldsymbol{e}$) and generalised velocities (flows, $\boldsymbol{f}$).

Hence, bond-graph principles are too restrictive and should therefore not be used in the design of a sequence-control layer, unless bond graphs are required for interfacing with the loop-control layer. Instead, an alternative sequence-control interface is needed with which energy exchange can be calculated.

An observation that can be made from Sections 2.2 and 3.1 is that all control layers share common basic principles on interfacing. Namely, that any two control entities interfacing with each other share an input/output connection over which a command or feedback signal can be sent. Notice that 'command' and 'feedback' are generic terms that include any sort of transmitted control signal. Typically, at least 1 command and at least 1 feedback signal are communicated. This is illustrated in Figure 3.1. These principles are important to identify as they form the basis of control interfaces. They also exemplify the semantics of 'command' and 'feedback' used in this project to distinguish between control signals.

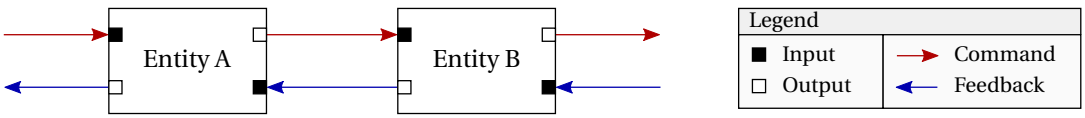


Figure 3.1: Control signals can be of type 'command' (e.g. actuation torques) or of type 'feedback' (e.g. measured velocities). A control signal that is an output of entity A, is an input of entity B, and vice versa.

Since energy exchange can be calculated with Equation A.2 using the four physical variables shown in Tables A.1 and A.2, the interface should be able to communicate these four variables. If either $\boldsymbol{e}$ or $\boldsymbol{p}$ together with either $\boldsymbol{f}$ or $\boldsymbol{q}$ are known, are collocated (Section 3.1), and are belonging to the same physical domain, then energy exchange can be computed by a control system. Therefore, the control interface of an energy-aware sequence-control layer should consist of command/feedback type signals of which one signal type can transmit a time-dependent $\boldsymbol{e}$-$\boldsymbol{p}$ pair or a time-dependent $\boldsymbol{f}$-$\boldsymbol{q}$ pair. The difference between this proposed control interface and bond graphs is illustrated in Table 3.1, which shows the 8 possible combinations of physical variables with which energy exchange can be calculated. Computing energy exchanges from these combinations is treated in Section 3.3.

Table 3.1: Possible combinations of physical variables in a bond graph compared to the proposed control interface for energy-aware sequence control.

	Bond Graph		**Proposed Control Interface**	
#	Command	Feedback	Command	Feedback
1.	$\boldsymbol{e}$	$\boldsymbol{f}$	$\boldsymbol{e}$	$\boldsymbol{f}$
2.			$\boldsymbol{e}$	$\boldsymbol{q}$
3.			$\boldsymbol{p}$	$\boldsymbol{f}$
4.			$\boldsymbol{p}$	$\boldsymbol{q}$
5.	$\boldsymbol{f}$	$\boldsymbol{e}$	$\boldsymbol{f}$	$\boldsymbol{e}$
6.			$\boldsymbol{f}$	$\boldsymbol{p}$
7.			$\boldsymbol{q}$	$\boldsymbol{e}$
8.			$\boldsymbol{q}$	$\boldsymbol{p}$

3.2.2 Energy Interface

The control interface proposed in Section 3.2.1 would be sufficient for developing a non-energy-aware sequence controller. However, for the development of a sequence controller that is energy-aware, the control interface on its own does not suffice. An additional interface is needed to provide an energy-communication standard for the sequence-control layer, that is, an energy interface is needed. Whilst defining this interface, good practises in design (Section 2.1) are kept in mind. In order to be generically compliant to other (past and future) work, the energy interface should incorporate energy variables that are commonly used in robotic applications. Below follows an analysis that identifies such energy variables.

Franken (2011) and Brodskiy (2014) worked on energy in robotics (described in Section 2.4) and made use of 'Energy Transfer' within their systems to transfer energies to, from, and between the energy tanks of passivity layers. An example of one such energy transfer is the supply of energy budgets to energy tanks. In Section 1.1 it is stated that "energy awareness means to plan and perform actions taking into account a system's energetic state". The energy-tank level is a state variable of passivity layers and can be used to base the planning and performing of control actions on. Therefore, it should be possible to communicate the energy-tank level from a passivity layer to other control components. Hence, the first two variables that should be incorporated in the energy interface for an energy-aware sequence-control layer are (1) E_{transfer} and (2) E_{tank}. These two energy variables are mainly used for passive control; they are virtual energies that exist inside the controller.

There exist more energy variables to be considered in a control strategy. Since energy is often split up in a kinetic and a potential component, these are added as generic variables to the energy interface; (3) E_{kinetic} and (4) $E_{\text{potential}}$. The kinetic energy in a robotic control system corresponds to robot-mass movement. The potential energy in a robotic system relates to potential energy in the actuators and gravity:

- Passive loop controllers at RaM often model a virtual spring 'attached' between the robot's end effector (e.g. a gripper) and a setpoint (desired end-effector position and orientation) to determine actuator torques. The potential energy contained within an elastic element (such as a virtual spring) could contain useful information to take into account by a controller deciding on a control action. Hence, another variable for the energy interface is: (5) E_{elastic}.

- Furthermore, robot controllers often apply active gravity compensation. To allow the possibility of including gravitational-energy information in a control strategy, energy variable (6) $E_{\text{gravitational}}$ is added to the energy interface.

Finally, to also allow the communication of a sum of energies, (7) E_{total} is added to the energy interface. These 7 energy variables are expected to provide sufficient energy-communication possibilities between components in any energy-aware sequence control layer. Figure 3.2 illustrates an example of the 7 identified energy variables in a robotic control systems. Be aware that these energy classifications are intended to contain specific information to be used in a control strategy and that, in real-world physics, energy does not stick to such classifications.

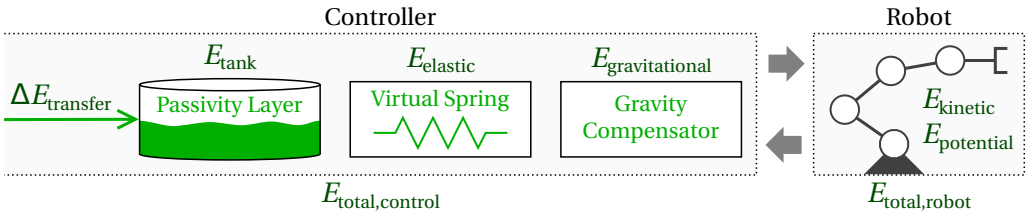


Figure 3.2: Energy classifications in a robotic control system that can be useful in the control strategy of an energy-aware sequence controller.

Unlike signals in the control interface proposed in Section 3.2.2, energies are not commanded or fed back. Instead, energy signals are scalars that are simply communicated or—depending on their sign—added or subtracted. Hence, besides an input/output configuration, further distinction of signal types (e.g. 'command' and 'feedback') is not needed for an energy interface. Summarising, the energy interface of an energy-aware sequence-control layer should at least incorporate the 7 identified scalar energy variables listed in Table 3.2.

Table 3.2: Proposed energy interface consisting of 7 energy variables to be used in an energy-aware sequence-control layer.

#	Proposed Energy Interface	Description
1.	E_{transfer}	energy transferred between energetic components
2.	E_{tank}	passivity layer's energy-tank level
3.	E_{kinetic}	kinetic energy
4.	$E_{\text{potential}}$	potential energy
5.	E_{elastic}	energy in an elastic element (e.g. a virtual spring)
6.	$E_{\text{gravitational}}$	energy due to gravity
7.	E_{total}	sum of energies in a (sub)system or component

Note that, out of the 7 energy classifications, all except the first (E_{transfer}) relate to an entity's energetic state. Hence, E_{transfer} can be used for the transmission of energies that are to be added or subtracted, for example supplying energy budgets to a passivity layer's energy tank. The other energy variables can be used to transmit information about an energy state, for example updating a controller about the energy contained by a virtual spring using E_{elastic}.

3.3 Computing Energy Exchange

Modern controllers are digital and require discrete-time equations for the computation of energy exchanges. To compute energies from physical interactions, the corresponding equations need to be compliant to the control interface of Section 3.2.1. Inserting the physical-variable combinations of Table 3.1 into Equation A.2 and using the time-dependent relations shown in

the tetrahedron of state in Figure A.1 yields four possible combinations to calculate energy flow with[3]:

$$\frac{\mathrm{d}E}{\mathrm{d}t} = \boldsymbol{e}^\top \cdot \boldsymbol{f} \qquad \Rightarrow \qquad \mathrm{d}E = \boldsymbol{e}^\top \cdot \boldsymbol{f} \cdot \mathrm{d}t \tag{3.1}$$

$$= \boldsymbol{e}^\top \cdot \frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} \cdot \mathrm{d}t \quad = \boldsymbol{e}^\top \cdot \mathrm{d}\boldsymbol{q} \tag{3.2}$$

$$= \frac{\mathrm{d}\boldsymbol{p}^\top}{\mathrm{d}t} \cdot \boldsymbol{f} \cdot \mathrm{d}t \quad = \mathrm{d}\boldsymbol{p}^\top \cdot \boldsymbol{f} \tag{3.3}$$

$$= \frac{\mathrm{d}\boldsymbol{p}^\top}{\mathrm{d}t} \cdot \frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} \cdot \mathrm{d}t = \mathrm{d}\boldsymbol{p}^\top \cdot \mathrm{d}\boldsymbol{q} \cdot \frac{1}{\mathrm{d}t}. \tag{3.4}$$

A digital controller updates its signals at fixed time intervals. Consider the current sample time of a controller to be t_k. The previous sample time is t_{k-1} and the next (future) sample time is t_{k+1}. During a sample interval (in between two subsequent sample times), the discrete-time controller is idle and no new data is obtained. Hence, the command signal outputted by a digital controller is constant during a sample interval. This is not the case for the sampled feedback signal, which is variable as illustrated in Figure 3.3.

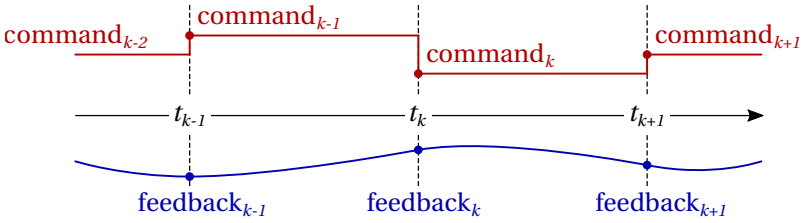


Figure 3.3: During sample intervals, a digital controller's command signal is constant whereas the sampled feedback signal varies.

Figure 3.3 shows that, throughout the latest sample interval, $[t_{k-1}, t_k]$, the value of the command signal, command$_{k-1}$, is known to the digital controller, but that this is not the case for the varying feedback signal, whose intermediate value(s) would have to be estimated. A direct consequence is that Equations 3.1 to 3.4 are rewritten into 8 (rather than 4) discrete-time equations for a digital controller to compute the amount of exchanged energy, ΔE_k, during sample interval $[t_{k-1}, t_k]$. These 8 discrete-time equations correspond to the 8 command/feedback combinations listed in Table 3.1 and can be represented as follows:

$$\Delta E_k = E_k - E_{k-1} = \begin{cases} \boldsymbol{e}_{k-1}{}^\top \cdot \frac{1}{2}\left(\boldsymbol{f}_k + \boldsymbol{f}_{k-1}\right) \cdot \Delta t_k & \text{command: } \boldsymbol{e}, \text{ feedback: } \boldsymbol{f} \\ \frac{1}{2}\left(\boldsymbol{e}_k + \boldsymbol{e}_{k-1}\right)^\top \cdot \boldsymbol{f}_{k-1} \cdot \Delta t_k & \text{command: } \boldsymbol{f}, \text{ feedback: } \boldsymbol{e} \\ \boldsymbol{e}_{k-1}{}^\top \cdot \left(\boldsymbol{q}_k - \boldsymbol{q}_{k-1}\right) & \text{command: } \boldsymbol{e}, \text{ feedback: } \boldsymbol{q} \\ \boldsymbol{e}_k{}^\top \cdot \left(\boldsymbol{q}_k - \boldsymbol{q}_{k-1}\right) & \text{command: } \boldsymbol{q}, \text{ feedback: } \boldsymbol{e} \\ \left(\boldsymbol{p}_k - \boldsymbol{p}_{k-1}\right)^\top \cdot \boldsymbol{f}_k & \text{command: } \boldsymbol{p}, \text{ feedback: } \boldsymbol{f} \\ \left(\boldsymbol{p}_k - \boldsymbol{p}_{k-1}\right)^\top \cdot \boldsymbol{f}_{k-1} & \text{command: } \boldsymbol{f}, \text{ feedback: } \boldsymbol{p} \\ \left(\boldsymbol{p}_k - \boldsymbol{p}_{k-1}\right)^\top \cdot \left(\boldsymbol{q}_k - \boldsymbol{q}_{k-1}\right) \cdot \frac{1}{\Delta t_k} & \text{command: } \boldsymbol{p}, \text{ feedback: } \boldsymbol{q} \\ \left(\boldsymbol{p}_k - \boldsymbol{p}_{k-1}\right)^\top \cdot \left(\boldsymbol{q}_k - \boldsymbol{q}_{k-1}\right) \cdot \frac{1}{\Delta t_k} & \text{command: } \boldsymbol{q}, \text{ feedback: } \boldsymbol{p} \end{cases}. \tag{3.5}$$

[3]Note that these four equations implicitly assume linearity if interval dt is not infinitesimal.

Herein, the time duration between current sample time t_k and previous sample time t_{k-1} is represented by $\Delta t_k = t_k - t_{k-1}$ and (for the reason explained above) in the [command, feedback] combinations of $[\boldsymbol{e}, \boldsymbol{f}]$ and $[\boldsymbol{f}, \boldsymbol{e}]$ an average value of the feedback signal is taken. Equation 3.5 can be sufficiently accurate for digital controllers that use short sample intervals. However, if higher accuracy is needed (as could be the case for systems with longer sample intervals), higher order approximations of the feedback signal during a sample interval could be applied.

Equation 3.5 implies time-discreteness, but not amplitude-discreteness of the sampled command and feedback signals. Figure 3.3 shows that the feedback signal is assumed amplitude-continuous. Amplitude continuity should also hold for the command signal, because if it would be amplitude-discrete, Equation 3.5 would result in the computation of amplitude-discrete energy exchanges—which would *not* be physics-conformal. Hence, the command signal outputted by an energy-aware controller can be discrete in time, but must be continuous in amplitude.

3.3.1 An Exception for Screw Theory

Equation 3.5 can be used to compute energy exchanges from any of the variables listed in Tables A.1 and A.2. In order to yield accurate and correct computations, these variables must (as stated above) be collocated and belonging to the same domain. However, from Table A.2 it can be seen that there exists one exception to which Equation 3.5 cannot be applied. This exception occurs for the screw theory domain, where the generalised displacement ($\boldsymbol{q}$) is an H-matrix of size 4×4 whereas the generalised force ($\boldsymbol{e}$), generalised momentum ($\boldsymbol{p}$), and generalised velocity ($\boldsymbol{f}$) are of size 6×1. Hence, the problem is an impossible product between a matrix and a vector of mismatching dimensions. A solution is to calculate a twist matrix from the H-matrix, extract a twist vector, and use that twist vector in Equation 3.5 instead.

Equations that follow below consider H-matrices and twists of the robot's end-effector frame (Ψ_{ee}) with respect to and expressed in the inertial origin frame (Ψ_0). The placement of these frames on a robot is illustrated in Figure 3.4.

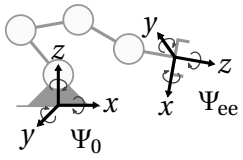


Figure 3.4: Illustrative placement of the inertial origin frame Ψ_0 and robot end-effector frame Ψ_{ee}.

Since $\boldsymbol{H}_0^{\text{ee}} = \left(\boldsymbol{H}_{\text{ee}}^0\right)^{-1}$, a twist matrix $\tilde{\boldsymbol{T}}_{\text{ee}}^{0,0} \in \mathbb{R}^{4\times4}$ can be calculated from H-matrix $\boldsymbol{H}_{\text{ee}}^0 \in \mathbb{R}^{4\times4}$ using Equation A.1:

$$\tilde{\boldsymbol{T}}_{\text{ee}}^{0,0} = \dot{\boldsymbol{H}}_{\text{ee}}^0 \cdot \boldsymbol{H}_0^{\text{ee}}. \tag{3.6}$$

The time derivative of $\boldsymbol{H}_{\text{ee}}^0$ between sample times t_k and t_{k-1} can be computed as follows:

$$\dot{\boldsymbol{H}}_{\text{ee}}^0(t_k) = \frac{\boldsymbol{H}_{\text{ee}}^0(t_k) - \boldsymbol{H}_{\text{ee}}^0(t_{k-1})}{\Delta t_k}. \tag{3.7}$$

A twist matrix $\tilde{\boldsymbol{T}}_{\text{ee}}^{0,0}(t_k)$ in sample interval $[t_{k-1}, t_k]$ can be estimated using Equations 3.6 and 3.7 and the average value of $\boldsymbol{H}_0^{\text{ee}}$ in that interval:

$$\tilde{\boldsymbol{T}}_{\text{ee}}^{0,0}(t_k) = \frac{\boldsymbol{H}_{\text{ee}}^0(t_k) - \boldsymbol{H}_{\text{ee}}^0(t_{k-1})}{\Delta t_k} \cdot \frac{\boldsymbol{H}_0^{\text{ee}}(t_{k-1}) + \boldsymbol{H}_0^{\text{ee}}(t_k)}{2}. \tag{3.8}$$

From Equations 3.8, B.1 and B.2, it can be seen that twist vector $\boldsymbol{T}_{\text{ee}}^{0,0} \in \mathbb{R}^{6\times1}$ is directly extractable from twist matrix $\tilde{\boldsymbol{T}}_{\text{ee}}^{0,0} \in \mathbb{R}^{4\times4}$. Thus, instead of H-matrix $\boldsymbol{H}_{\text{ee}}^0$ as a generalised displacement

($\boldsymbol{q}$), twist vector $\boldsymbol{T}_{\text{ee}}^{0,0}$ can be used as a flow ($\boldsymbol{f}$) in Equation 3.5. If a more accurate estimation of $\tilde{\boldsymbol{T}}_{\text{ee}}^{0,0}(t_k)$ in sample interval $[t_{k-1}, t_k]$ is needed, higher order approximations could be used for Equation 3.7 and the average of $\boldsymbol{H}_0^{\text{ee}}$ in Equation 3.8.

3.4 Use Cases for Evaluating Energy Awareness

To allow the integration of energy awareness into a sequence controller to be evaluated, suitable evaluation scenarios are needed. From Section 3.1 it can be observed that there exist many scenarios in which energy awareness in sequence control could be useful. Section 1.1 states the example of a robotic manipulator that physically interacts with an unknown environment. Unexpected physical interactions can become the cause of a fault in a robotic system. Such a fault can activate an error, which can propagate into a failure, which can cause another fault, etc. This fault propagation process was presented by Avizienis et al. (2004) as shown in Figure 3.5.

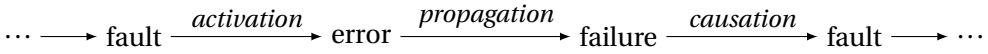


Figure 3.5: Fault propagation process (Avizienis et al., 2004).

Figure 3.5 shows a process that is to be prevented. An energy-aware sequence controller should be able to detect a wide range of faults and errors by assessing the energetic state of the system. After fault or error detection, it should plan and execute a correction approach in order to prevent any further failures. Carlson et al. (2004) identified the following physical failure classifications: Effector, Sensor, Control system, Power, and Communications. An energy-aware sequence controller may be able to correct faults and errors in these fields, except for the field of Power as this lies beyond the controller's reach. The remaining fault classifications can be extended to fault scenarios. The following generic fault scenarios, each of which requires a different correction approach, are proposed:

I. Effector: an unexpected effector (e.g. obstacle) along the trajectory towards the target.

II. Sensor: a sensor or actuator failure.

III. Control system: incorrect target specifications (e.g. dimensions and/or weight).

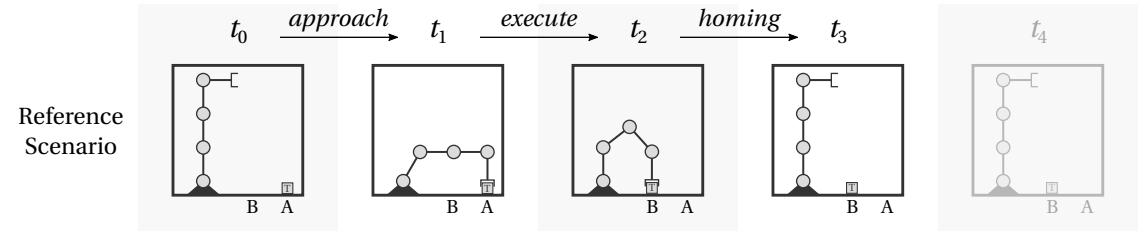
IV. Control system: an incorrect target pose (position and/or orientation).

V. Communication: communication delay or loss.

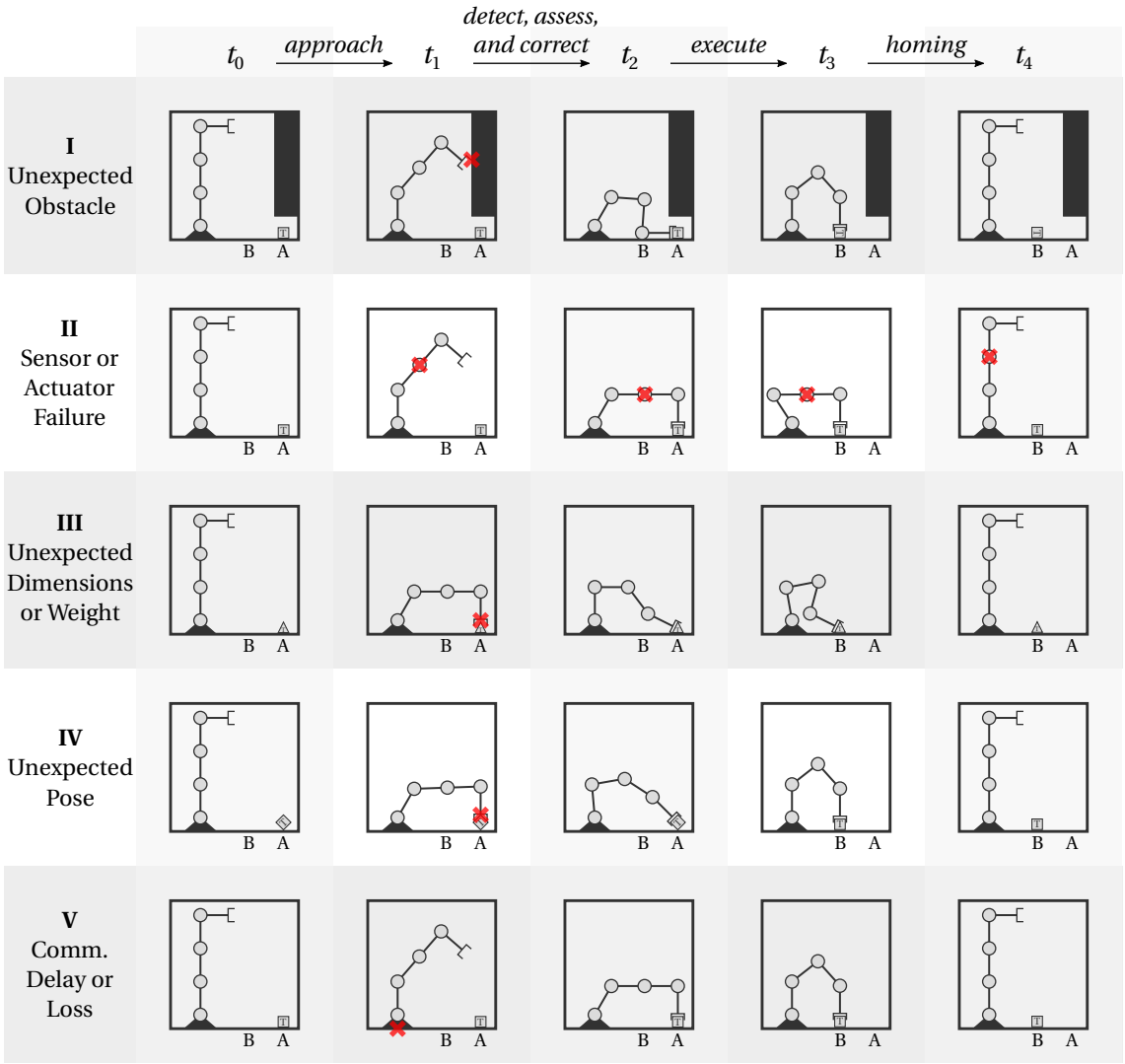
These fault-scenarios can be used as use cases in which an energy-aware sequence controller uses energy information to find a suitable correction approach. The proposed fault-scenario use cases and examples of corresponding robot correction approaches are illustrated in Figure 3.6. The presented use cases are generic in the sense that they can be adapted to various situations; the robotic manipulator can be replaced by any other robot and the target object can be replaced by another task. Figure 3.6b presents four main sequences for each fault scenario:

1. Interval $[t_0–t_1]$: approaching the target object.
2. Interval $[t_1–t_2]$: fault occurrence, detection, assessment, and approach correction.
3. Interval $[t_2–t_3]$: executing the task; moving the target object from A to B.
4. Interval $[t_3–t_4]$: returning to home pose.

Note that the second sequence (fault occurrence, detection, assessment, and approach correction) does not take place under normal operations, as illustrated by the reference scenario in Figure 3.6a. Any of the fault-scenario use cases presented in Figure 3.6 may be used to evaluate energy awareness of a controller with, by letting the controller plan and execute a correction approach based on energy information in the system.



(a) Reference scenario without faults.



(b) Five fault scenarios.

Figure 3.6: Fault scenarios with a robotic manipulator tasked to move a target object from location A to location B. Each scenario requires a different correction approach to prevent system failure and successfully execute the task.

3.5 Requirements for Design and Implementation

From the information in Chapter 2 and the analyses in this chapter, requirements for the design and implementation of an energy-aware sequence-control architecture have been established. The requirements are prioritised following the MoSCoW (Must, Should, Could, Will not) criteria. Prioritisation is based on the following question: “is this requirement necessary in energy-aware sequence control?”

3.5.1 Design Requirements

The design of an energy-aware sequence-control layer ...

1) ... *should* be based on good practises in design (Section 2.1).
2) ... *should* incorporate a connection point for task commands coming from the supervisory-control layer (Section 2.2).
3) ... *could* incorporate a connection point for sensor data coming from the plant (Section 2.2).
4) ... *must* receive state feedback from the loop-control layer (Section 2.2).
5) ... *must* send amplitude-continuous setpoint commands, that are adjusted based on energy feedback, to the loop-control layer (Sections 2.2, 3.1 and 3.3).
6) ... *must* be passive (Sections 2.4 and 3.1).
   a) ... *should* incorporate a passivity layer (Section 3.1).
      i) ... *could* reuse (reusing is a good practise in design—Section 2.1) the design of a passivity layer from some previous work (Section 3.1).
   b) ... *should* incorporate accurate energy-budget estimations (Section 2.4).
7) ... *must* incorporate a control interface consisting of command and feedback inputs and outputs (illustrated in Figure 3.1) that can communicate the four physical variables as indicated in Table 3.1 (Section 3.2.1).
   a) ... *should* not (!) make use of bond graphs, except possibly for connecting to the loop-control layer (Section 3.2.1).
      i) ... *should* incorporate both impedance and admittance functionality if bond graphs are used (Section 3.2.1).
8) ... *must* incorporate an energy interface for standardised communication of energy data as indicated by Table 3.2 (Section 3.2.2).
   a) ... *should* be compliant to the plan within RaM to eventually add an energy supervisor (Section 2.4).

3.5.2 Implementation Requirements

Note: all design requirements listed in Section 3.5.1 also apply to the implementation.

The implementation of an energy-aware sequence-control layer ...

1) ... *must* provide real-time guarantees and operate in hard- and/or soft real-time (Section 2.2).
2) ... *could* be compared to other implementations of sequence control, such as those analysed in Section 3.1, on the basis of system-level properties like safety, dependability and performance (Section 3.1).
3) ... *should* show improved system-level properties due to energy awareness (Section 3.1).
4) ... *must* use sensors and actuators that are accurately calibrated (Section 3.1).
5) ... *must* use collocated (both in time and positioning) actuation commands and measurement feedback data (Section 3.1).
6) ... *should* be able to demonstrate its energy awareness (Section 3.4).
   a) ... *could* include any of the use cases presented in Figure 3.6 to demonstrate energy awareness (Section 3.4).

4 Design and Implementation

This chapter treats the project's subgoal of "designing and implementing the architecture of an energy-aware sequence-control layer", stated in Section 1.3. The architecture design and subsequent conceptual implementation have been based on the design and implementation requirements listed in Section 3.5.

4.1 Design of an Energy-aware Sequence-Control Architecture

The architecture design is to form a basis for energy-aware sequence control by providing a systematic approach on taking into account the system's energetic state in the control strategy. The design comprises a network of components and subcomponents, which, for clarity and understandability, is presented in steps of increasing detail.

4.1.1 Interface Designs

Before presenting the components that make up the Sequence-Control layer, the interface designs are explained. The interfaces cover the 5Cs-concern of Communication between control entities (Design Requirement 1).

Figure 4.1 shows signal notations used in all design diagrams that follow in the remainder of this chapter. The arrows are colour coded to distinguish between the different types of signals. The red and blue arrows represent command- and feedback signals, respectively, and belong to the control interface (Design Requirement 7). The green arrows represent energy signals and belong to the energy interface (Design Requirement 8). The gray arrows represent other signals for which it is not in this project's scope to define an interface. These 'other' signals can contain any unspecified data, such as implementation-dependent state feedback data.

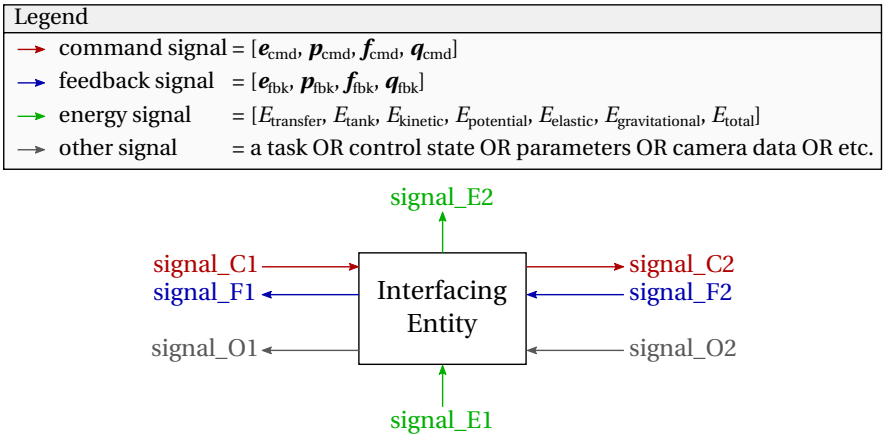


Figure 4.1: Design interpretation of interface signals. The command, feedback and energy arrows represent vector signals that contain multiple variables. The 'other' signals can contain any data. Names accompanying an arrow (e.g. signal_C1, signal_F1, signal_O1, signal_E1, etc.) are merely a label to distinguish the signal with, not its contents. The layout of signals in this diagram serves as an example.

The command (red) and feedback (blue) signals transmit arrays consisting of physical variables $\boldsymbol{e}$, $\boldsymbol{p}$, $\boldsymbol{f}$, and $\boldsymbol{q}$ that can contain any of the quantities listed in Tables A.1 and A.2 (Design Requirement 7). The control interface does not restrict using multiple variables concurrently, that is, a control signal (command or feedback) transmits all of the $\boldsymbol{e}$, $\boldsymbol{p}$, $\boldsymbol{f}$, and $\boldsymbol{q}$ variables at once and one, multiple, or all of them can be filled in. The reason for this design choice is to establish an interface that is implementation-agnostic (in line with Design Requirement 1). If, for example, the interface would be used in a control system par-

tially consisting of torque-controlled actuators and partially of position-controlled actuators, this control interface would allow such combinations to be commanded. It is left up to the developer to determine which signals to make use of and to make sure those signals comply to the rest of the system.

The energy signals (green) correspond to the energy interface and transmits an array of the variables listed in Table 3.2 (Design Requirement 8). As with the control interface, the energy interface also does not restrict using multiple variables concurrently, i.e. the energy signal enables any of the listed energy variables to be filled in. Hence, also here it is left up to the developer to decide which variables to make use of.

Design diagrams can also contain signal labels; a name accompanying a signal. The sole purpose of these labels is to distinguish and refer to specific signals, e.g. 'signal_E1' in Figure 4.1. This means that an energy label called 'tank', for example, would merely refer to that specific signal (an array of variables). It would not (!) be referring to the specific scalar variable E_{tank} contained within that vector. The colour of a signal (not its name/label) indicates the interface a signal belongs to and, hence, what its contents are.

4.1.2 Top View of the Layered Control System

Using the interface design of Section 4.1.1, a top view of the full control-system design is presented in Figure 4.2. It shows all of the main control layers and their interconnections in line with the control-system representations of Figures 2.2 and 2.3.

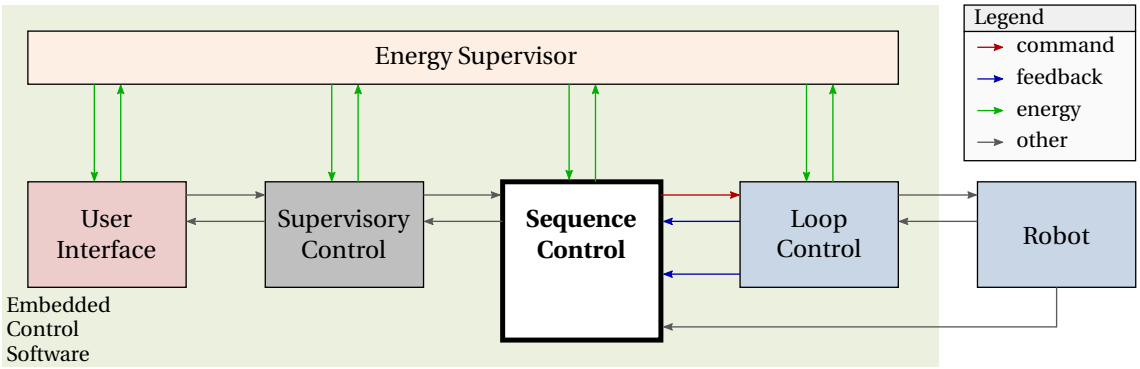


Figure 4.2: Design top view of Sequence Control within the layered control system.

Following Design Requirement 8a and using the energy interface, each layer communicates energy information with an Energy Supervisor. For example, Sequence Control receives energy information of Loop Control, not directly from the Loop Control layer itself, but via the Energy Supervisor. Using the control interface, Sequence Control communicates a command signal and a feedback signal with Loop Control (Design Requirement 4) and receives an additional feedback signal from Loop Control that contains state feedback of the Robot[1] (which Loop Control obtains from its communication with the Robot).

The signals between the User Interface, Supervisory Control and Sequence Control (Design Requirement 2), and the signals between Loop Control and the Robot, are classified as 'other' signals, because they are implementation-dependent and their specific definition is not in the scope of this project. The 'other' signal from the Robot to Sequence Control generically represents any sensor data (Design Requirement 3).

[1] In accordance with Section 3.4 and Figures 1.3 and 2.2, the Plant is considered to consist of a Robot (and its physical Environment).

4.1.3 Detailed Design of the Energy-aware Sequence-Control Layer

A step of increasing detail following the design's top view shown in Figure 4.2, is to show the main components inside the Sequence-Control layer and omit any layers that are not directly connected (i.e. the User Interface). This is shown in Figure C.2. Next, Figure 4.3 presents the Sequence-Control layer design in detail, leaving out all external entities and showing all relevant signal names.

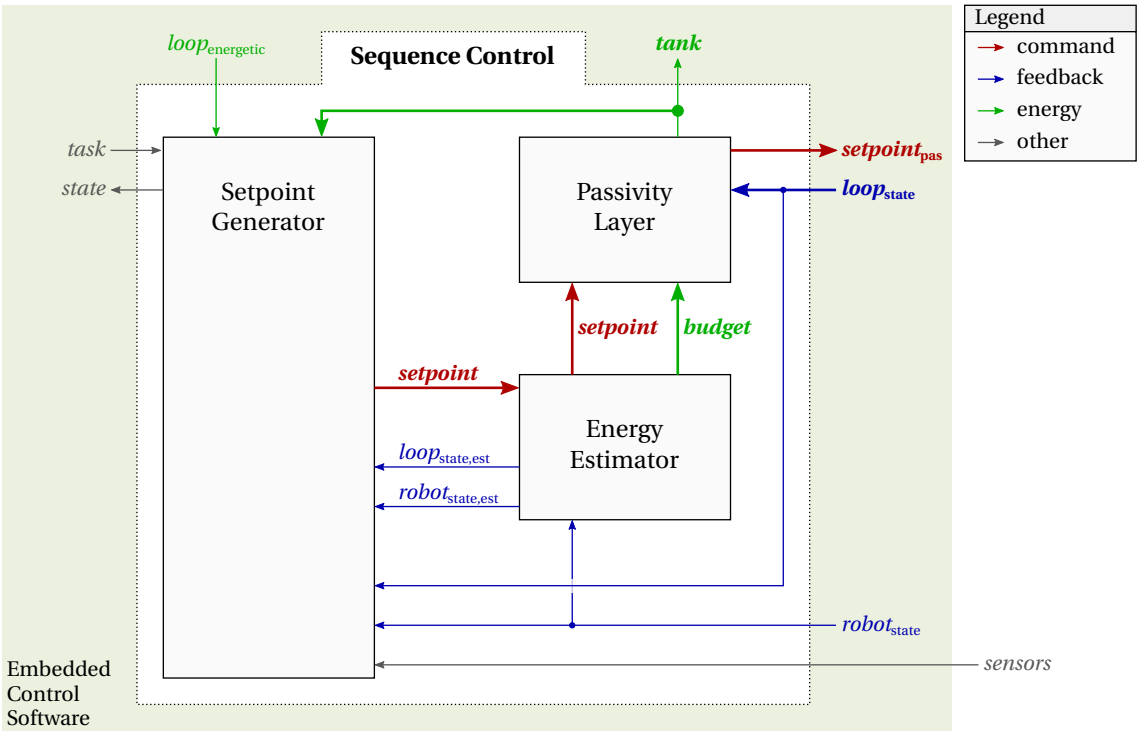


Figure 4.3: Detailed design of the Sequence-Control layer leaving out external entities and including signal names.

Applying component-based design (in line with Design Requirement 1), the designed Sequence-Control layer incorporates three main components in order to fulfil the remaining design requirements:

- a *Setpoint Generator* to satisfy Design Requirement 5,
- an *Energy Estimator* to satisfy Design Requirement 6b,
- and a *Passivity Layer* to satisfy Design Requirement 6a.

These three control components define the Sequence-Control layer's core functionality, covering its 5Cs-concern of Computation. With the Setpoint Generator, Energy Estimator and Passivity Layer, and the interface signals interconnecting them, Figure 4.3 also addresses the 5Cs-concern of Composition of the Sequence-Control layer (Design Requirement 1).

The Setpoint Generator calculates setpoints based on unspecified commands coming from the Supervisory-Control layer and feedback signals coming from the Energy Supervisor, Energy Estimator, Passivity Layer, Loop Control, and the Robot. The Energy Estimator provides energy budgets to the Passivity Layer's energy tank and also forwards the setpoint. The Passivity Layer provides a passive connection to Loop Control. The three components communicate the following input and output signals:

- Command signals (using the control interface):
  - *setpoint* contains the setpoint generated by the Setpoint Generator and is sent to the Energy Estimator. The Energy Estimator forwards this exact same signal to the Passivity Layer. *setpoint* is intended to control/steer the Loop Controller.
  - $setpoint_{\text{pas}}$ is a possibly attenuated version of the original *setpoint* signal intended to enforce passivity of the sequence-control layer. $setpoint_{\text{pas}}$ is outputted by the Passivity Layer to the Loop-Control layer.
- Feedback signals (using the control interface):
  - $loop_{\text{state}}$ contains physical variables of the Loop-Control layer and is transmitted from the Loop-Control layer to both the Passivity Layer and the Setpoint Generator.
  - $robot_{\text{state}}$ contains physical variables of the Robot and is transmitted from the Loop-Control layer to the Energy Estimator and Setpoint Generator. The Loop-Control layer obtains $robot_{\text{state}}$ from its power-continuous connection with the Robot.
  - $loop_{\text{state,est}}$ and $robot_{\text{state,est}}$ are model-based estimations of $loop_{\text{state}}$ and $robot_{\text{state}}$, respectively, and are provided by the Energy Estimator (explained in Section 4.1.5) to the Setpoint Generator.
- Energy signals (using the energy interface):
  - $loop_{\text{energetic}}$ contains the Loop-Control layer's energetic state communicated via the Energy Supervisor to the Setpoint Generator.
  - *budget* contains an energy budget estimated by the Energy Estimator for the Passivity Layer's energy tank.
  - *tank* contains the energy state of the Passivity Layer's energy tank (recall from Section 4.1.1 that an energy-interface signal always comprises the full array of energy variables). *tank* is outputted by the Passivity Layer to the Setpoint Generator and to the Energy Supervisor.
- Other signals:
  - *task* and *state* represent control signals between the Setpoint Generator and Supervisory-Control layer.
  - *sensors* is a feedback signal from the Robot to the Setpoint Generator that can contain any sensor data, possibly from multiple sensors (e.g. from cameras, lasers, IMUs, GPSes, etc.).

In the detailed design shown in Figure 4.3 and other design diagrams that follow in the remainder of this chapter, necessary signals—those that make up the bare minimum for *implementing* an energy-aware Sequence-Control layer—are highlighted by larger arrows with bold font names[2]. The other signals are of varying importance, but not considered absolutely necessary in the implementation of an energy-aware Sequence-Control layer. This distinction between necessary and unnecessary signals provides a guide (for developers) on which signals to focus on. It is important to take note that the designs shown in this chapter do not restrict the addition of new signals in future work, as long as they comply to the interfaces presented in Section 4.1.1.

Below follow explanations on component functionalities and on all signals, including explanations on why certain signals are deemed necessary and others are not. First, the Setpoint Generator and its signals are described in Section 4.1.4, followed by the Energy Estimator in Section 4.1.5, and then the Passivity Layer in Section 4.1.6.

[2]Not to be confused with bold-font notations of (mathematical) symbols used to indicate vectors and matrices.

4.1.4 Setpoint Generator Design

The Setpoint Generator's design is to satisfy Design Requirement 5. In Section 1.1, it is stated that "energy awareness means to plan and perform actions taking into account a system's energetic state". Accordingly, the Setpoint Generator is designed to output setpoints taking into account the system's energetic state by assessing the input energy signals *tank* and $loop_{\text{energetic}}$. The setpoints outputted by the Setpoint Generator are to steer the Loop Controller to a certain target. This target is communicated via the *task* signal coming from the Supervisory Controller. It is the Setpoint Generator's responsibility to output setpoints that make the Loop Controller reach its targets, that is, it should output the setpoints along a feasible trajectory towards the target. This is illustrated in Figure 4.4.

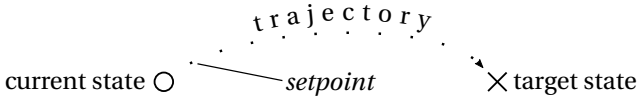


Figure 4.4: The Setpoint Generator commands setpoints along a trajectory to steer the Loop Controller from its current state towards a target state.

From the description above, two main operations can be distinguished for the Setpoint Generator: energy assessment and trajectory planning. These two operations are included in the Setpoint Generator design in the form of the Energy Assessing and Trajectory Planning blocks, as shown in Figure 4.5. The designed Setpoint Generator provides energy-guided control; it assesses energy states of the control system and adapts its trajectory planning and setpoint generation accordingly. Important to keep in mind is that subsequent setpoints must be amplitude-continuous (Design Requirement 5) and contained by at least one of the four physical variables $\boldsymbol{e}$, $\boldsymbol{p}$, $\boldsymbol{f}$, and $\boldsymbol{q}$.

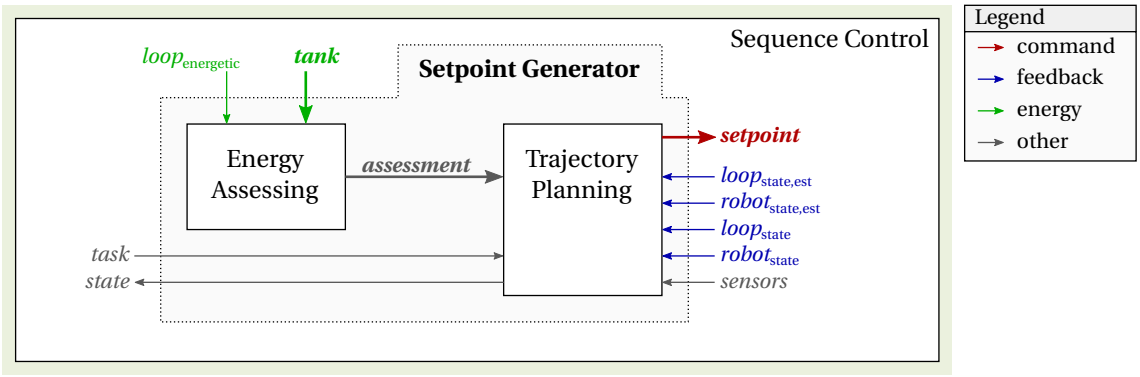


Figure 4.5: Setpoint Generator design consisting of an Energy Assessing block and a Trajectory Planning block.

The Energy Assessing block is used to assess energy states within the control system. To this end, it receives energy signals $loop_{\text{energetic}}$ and *tank*, assesses them, and forwards the assessment result (*assessment*) to the Trajectory Planning block. Depending on implementation, the Energy Assessing block can perform, for example, energy tank monitoring, fault and error detections (as described in Section 3.4), or detection and identification of physical interactions.

The Trajectory Planning block receives and processes all of the non-energy feedback signals and, based on these inputs, plans a trajectory and outputs setpoints along it. Situations may occur in which the Trajectory Planning block, through the Energy Assessment block, finds that problems occur along a certain trajectory. The Trajectory Planning block should then be able to divert onto an alternative trajectory in a new attempt to reach the target.

Necessary Implementation Signals

The Sequence-Control layer's commanded output, *setpoint*, is deemed necessary for the implementation of energy-aware sequence control following Design Requirement 5. Another signal deemed necessary is the *assessment* signal, because energy-aware trajectory planning is not achieved without it.

Under the assumption that a Loop Controller will reach its commanded setpoints, it is possible to develop a Setpoint Generator that generates setpoints regardless of control-feedback signals. Therefore, the control-feedback signals to the Setpoint Generator ($loop_{\text{state}}$, $robot_{\text{state}}$, $loop_{\text{state,est}}$, and $robot_{\text{state,est}}$) are not deemed absolutely necessary for implementing energy-aware Sequence Control. Despite not being absolutely necessary by design, the state signals $loop_{\text{state}}$, and $robot_{\text{state}}$ can be important feedback signals to base setpoint generation on, depending on implementation. The estimated signals $loop_{\text{state,est}}$ and $robot_{\text{state,est}}$ are supplementary feedback signals that have simply been added because they exist in the Energy Estimator. They may be implemented if desired.

Two energy signals are available to achieve energy awareness with: $loop_{\text{energetic}}$ and *tank*. The bare minimum to achieve energy awareness in the Sequence-Control layer is to implement only one of these two signals. *tank* contains the state of the Passivity Layer's energy tank and is available by definition of the Sequence-Control layer's design. $loop_{\text{energetic}}$ contains the energetic state of the loop controller, but may not be available depending on the loop controller's implementation. Because at least one of the two energy-feedback signals is required to enable energy awareness and availability can be guaranteed for *tank* and not for $loop_{\text{energetic}}$, *tank* is deemed necessary and $loop_{\text{energetic}}$ is not. This also attributes to the independence property of the Sequence-Control layer (Design Requirement 1). Nevertheless, $loop_{\text{energetic}}$ is considered an important feedback signal for assessing system energy properties closer related to the physical world and should be implemented if possible.

Other signals *sensors*, *task* and *state* are not deemed necessary in the implementation of energy-aware Sequence Control. The *sensors* feedback signal (not to be confused with feedback from the Robot to Loop-Control) is not necessary, because it is an optional signal that can be implemented in case sensor data is available. The signals *task* and *state* from and to the Supervisory-Control layer are unnecessary, because the Sequence-Control layer is designed to be independently functional (in accordance with Design Requirement 1), i.e. it can be implemented in a control system without higher control layers. Note that from the Supervisory-Control layer's perspective (if it would be present in the implementation), the *task* signal is necessary and the *state* signal may be necessary, depending on feedback requirements.

4.1.5 Energy Estimator Design

The Energy Estimator is inspired from the energy estimation block by Brodskiy (2014). Accordingly, it consists of real-time models of the Loop-Control layer, the Robot, and the Environment to determine model-based energy budgets for the Passivity Layer's energy tank. These energy budgets are an estimation of the amount of energy that the Loop-Control layer will require for the commanded *setpoint* signal. The energy estimator by Brodskiy (2014) obtained energy budgets from the amount of energy consumed (or extracted) by the Model of Robot. However, since the sequence-control layer commands a loop controller rather than a robot, the energy estimator here obtains its energy budgets from the amount of energy that the Model of Loop Control consumes to reach its setpoint. Note that this energy consumption also includes the amount of energy consumed by the Model of Robot and the Model of Environment. Figure 4.6 shows this in more detail.

Calculated energy budgets are sent to the Passivity Layer through energy-signal *budget*. Alongside *budget*, the Energy Estimator forwards necessary signal *setpoint* to the Passivity Layer. It

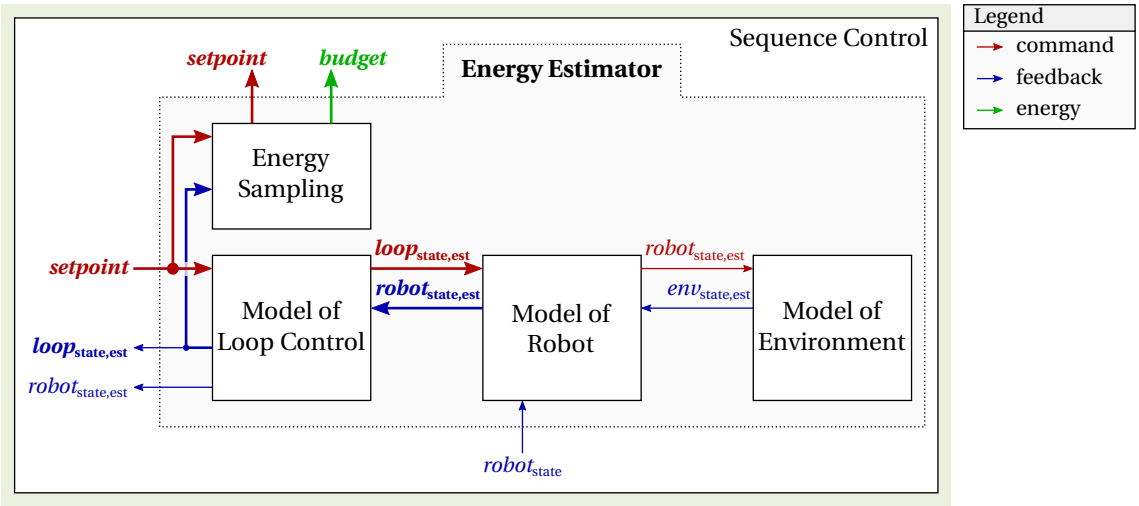


Figure 4.6: Energy Estimator design inspired by Brodskiy (2014) and consisting of a Model of Loop Control block, a Model of Robot block, a Model of Environment block, and an Energy Sampling block.

is important that *budget* and *setpoint* are two matching signals, i.e. that the energy budget has been estimated for that specific setpoint. This is why both should be sent at the same time by the Energy Estimator to the Passivity Layer. This means that if the Energy Estimator takes some time to estimate an Energy Budget and in the meantime a new setpoint has arrived, the Energy Estimator should wait with outputting the previous setpoint until its corresponding energy budget is ready. The Energy Estimator must not adjust *setpoint*, because the designed purpose is to only use *setpoint* as input and forward it together with a matching *budget* signal.

Figure 4.6 shows where the Energy Estimator's output feedback signals $loop_{\text{state,est}}$ and $robot_{\text{state,est}}$ (mentioned in Section 4.1.4) originate from. They are produced by the Model of Loop Control and Model of Robot blocks, respectively. Note that if the Loop-Control layer's source code is available, the Loop-Control layer can be accurately modelled by the Model of Loop Control block. Accurate modelling is important, because Brodskiy (2014) identified that inaccurate energy-budget estimations have a negative impact on system performance. Therefore, when implementing an Energy Estimator, during any trade-off involving accuracy, the importance of accurate budget estimations (and, hence, accurate models) for system performance should be kept in mind (Design Requirement 6b). To this end, as models are inadvertently imperfect, Model of Robot receives control-feedback signal $robot_{\text{state}}$ to counteract dynamic discrepancies between it and the real-world robot, i.e. to keep the Model of Robot synchronised with the physical robot.

Necessary Implementation Signals

Besides the necessary control-command signal, *setpoint*, the Energy Estimator receives synchronisation-signal $robot_{\text{state}}$, as explained above. Since an Energy Estimator could be developed without considering synchronisation, $robot_{\text{state}}$ is not deemed absolutely necessary in the implementation of energy-aware sequence control.

Based on signals *setpoint* and $loop_{\text{state,est}}$, the Energy Sampling block calculates an energy budget for the Passivity Layer's energy tank. Energy budgets are computed with Equation 3.5 and sent via energy-signal *budget*. This signal is deemed necessary, because the Passivity Layer cannot function without energy being provided to its energy tank. Since *budget* is necessary and depends on signals *setpoint* and $loop_{\text{state,est}}$, it is necessary to send the latter two signals to the Energy Sampling block in the implementation of an energy-aware Sequence-Control layer. However, despite being necessary internally, $loop_{\text{state,est}}$ is not necessary as an output signal of the Energy Estimator, as explained in Section 4.1.4.

Because the Model of Loop Control's output, $loop_{\text{state,est}}$, is a necessary signal to the Energy Sampling block and the behaviour of the Model of Loop Control depends on the behaviour of the Model of Robot, the interaction between these two blocks is also necessary. Hence, the signals $loop_{\text{state,est}}$ and $robot_{\text{state,est}}$ in between the Model of Loop Control and Model of Robot blocks are necessary in the implementation of energy-aware sequence control. It is possible to develop a Model of Robot without considering its environment, i.e. without a Model of Environment. Therefore, the signals $robot_{\text{state,est}}$ and $env_{\text{state,est}}$ are not considered absolutely necessary in the implementation of energy-aware sequence control.

4.1.6 Passivity Layer Design

Like the Energy Estimator presented in Section 4.1.5, the Passivity Layer is also inspired by the work of Brodskiy (2014) (Design Requirement 6ai). The Passivity Layer consists of an Energy Tank block, a Passive Zero-Order Hold (ZOH) block, and an Energy Sampling block, as shown in Figure 4.7. Brodskiy (2014) also included an Energy Transfer block in the Passivity Layer design, but since its only purpose here would be to add energy budgets to the energy tank, this functionality is included in the Energy Tank block. Encouraging reuse in the implementation (Design Requirement 1); the same Energy Sampling block can be implemented in the Passivity Layer of Figure 4.7 and the Energy Estimator of Figure 4.6.

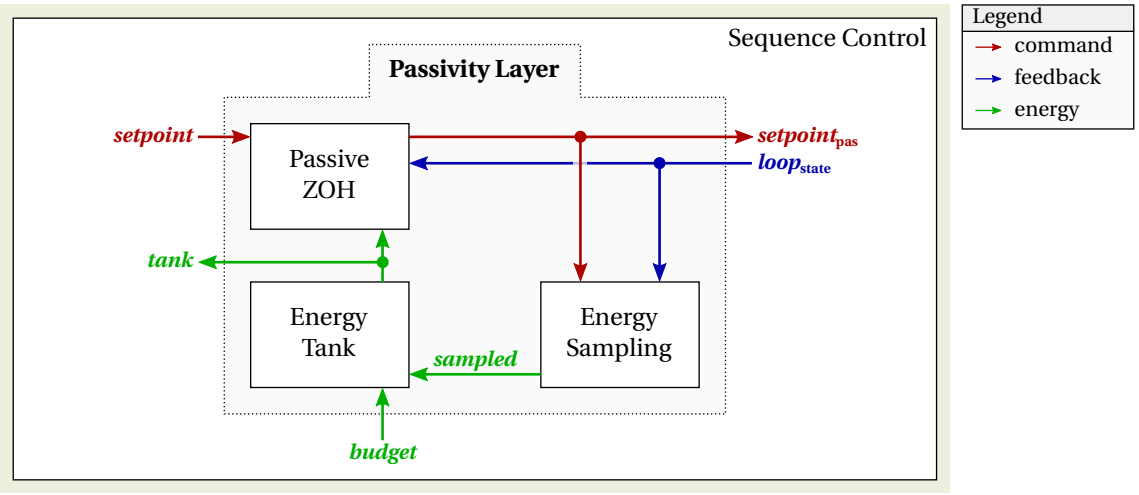


Figure 4.7: Passivity Layer design inspired by Brodskiy (2014), consisting of a Passive Zero-Order Hold (ZOH) block, an Energy Sampling block, and an Energy Tank block.

The Passivity Layer enforces passivity of the Sequence-Control layer by making sure the energy-tank level remains positive. This is achieved by the Passive ZOH block, which modulates the setpoint signal if need be. It estimates the amount of energy required for the upcoming sample interval $[t_k, t_{k+1}]$ (see Figure 3.3) and, if this energy estimate exceeds the current energy-tank level, attenuates the setpoint accordingly.

The energy-tank level estimated[3] for the next sample time ($E^*_{\text{tank},k+1}$) is calculated as a function of the current energy-tank level ($E_{\text{tank},k}$) and the estimated amount of energy required by the Loop-Control layer during the upcoming sample interval ($\Delta E^*_{\text{req},k+1}$):

$$E^*_{\text{tank},k+1} = E_{\text{tank},k} - \Delta E^*_{\text{req},k+1}. \tag{4.1}$$

This is a 'pessimistic' energy tank estimation as it excludes energy-budget supply, i.e. assumes a worst-case energy budget of 0 J. Using a pessimistic approach reduces the chance of breaking

[3] An asterisk (*) is used to mark variables that are estimated.

passivity due to uncertainty in the value estimation of $\Delta E^*_{\text{req},k+1}$. By taking Equation 3.5 and using a sample interval of $[t_k, t_{k+1}]$ instead of $[t_{k-1}, t_k]$, the estimated required energy exchange, $\Delta E^*_{\text{req},k+1}$, can be estimated with physical-variable estimations ($\boldsymbol{e}^*_{k+1}$, $\boldsymbol{p}^*_{k+1}$, $\boldsymbol{f}^*_{k+1}$, or $\boldsymbol{q}^*_{k+1}$). Estimation of physical variables is left up to the developer and could, for example, be based on a first-, or higher-order approximation.

Equation 4.1 shows that estimated energy-tank level, $E^*_{\text{tank},k+1}$, is lower than $E_{\text{tank},k}$ if the energy estimated to be required by the Loop Control layer, $\Delta E^*_{\text{req},k+1}$, has a positive value. It is important to note that Equation 3.5 does not exclude the possibility of $\Delta E^*_{\text{req},k+1}$ having a negative value. This would occur when the command and feedback variables are of opposing signs. In such a case, the Loop-Control layer would return energy into the Passivity Layer's energy tank, leading to an increase—rather than a decrease—of E_{tank} in Equation 4.1.

Depending on the amount of energy that is estimated to remain in the the energy tank, $E^*_{\text{tank},k+1}$, the Passive ZOH block either outputs the original setpoint command, *setpoint*, or an an attenuated version of it, $setpoint_{\text{att}}$:

$$setpoint_{\text{pas},k} = \begin{cases} setpoint_k & E^*_{\text{tank},k+1} \geq 0 \\ setpoint_{\text{att},k} & E^*_{\text{tank},k+1} < 0 \end{cases}. \tag{4.2}$$

The attenuated setpoint command, $setpoint_{\text{att}}$, is defined by Equation B.4 in such a way that Equation 4.1 results in $E^*_{\text{tank},k+1} = 0$ if $E_{\text{tank},k} > 0$ or in $E^*_{\text{tank},k+1} = E_{\text{tank},k}$ if $E_{\text{tank},k} \leq 0$.

The Energy Sampling block calculates the real exchanged energy by sampling the control signals $setpoint_{\text{pas}}$ and $loop_{\text{state}}$ using Equation 3.5. It writes the sampled energy to the Energy Tank block through energy-signal *sampled*. The Energy Tank block calculates the energy tank's level by adding the energy budget provided by energy-signal *budget* and subtracting the sampled energy provided by *sampled*.

Necessary Implementation Signals

As is explained in Sections 4.1.4 and 4.1.5, the signals *setpoint* and *budget* are necessary for the implementation of an energy-aware Sequence-Control layer. Since *budget* is directly dependent on energy-signal *sampled*, *sampled* is necessary as well. Also treated in Section 4.1.4 is energy-signal *tank*, which contains the energetic state of the energy tank. It is sent to the Setpoint Generator and to the Energy Supervisor. As explained in Section 4.1.4, the branch of *tank* that is sent to the Setpoint Generator is a necessary signal. On the other hand, the branch of *tank* that is sent to the Energy Supervisor is not necessarily implemented in energy-aware Sequence Control, because Sequence Control itself can function without this output signal. However, given that *tank* is the only energy output from the Sequence-Control layer to the Energy Supervisor and the Energy Supervisor is to supervise all control layers, *tank* is a necessary signal if an Energy Supervisor is implemented.

The Passivity Layer communicates two more signals: command-signal $setpoint_{\text{pas}}$ (written to the Loop-Control layer) and state-feedback signal $loop_{\text{state}}$ (read from the Loop-Control layer). Both are considered necessary in implementing energy-aware Sequence Control, because the Passivity Layer uses them to calculate the amount of energy it exchanges with the Loop-Control layer with. Both signals are used to satisfy Design Requirement 6 and $setpoint_{\text{pas}}$ is also used to satisfy Design Requirement 5.

4.2 Conceptual Implementation of an Energy-aware Sequence-Control Layer

In line with the implementation requirements posed in Section 3.5.2, this section presents a conceptual implementation of the energy-aware sequence-control layer design presented in Section 4.1. The implementation serves a proof of concept. It is intended to form a basis on implementing energy-aware sequence control rather than being a fully finished implementation with extensive functionalities. Throughout the implementation, good programming practises presented by Deitel and Deitel (2017) have been followed to comply to Design Requirement 1.

This section starts by explaining the loop controller and robot that have been implemented and the implications they pose on the sequence-control implementation. Then, forming the basis of the sequence-control implementation, an explanation on the implemented control- and energy interfaces is given. Hereafter, the overall functionality of the sequence-control layer is described, followed by detailed explanations on the implemented setpoint generator, energy estimator, and passivity layer components.

4.2.1 Loop Control and Robot

Implementation Requirement 3 specifies that "the implementation of an energy-aware sequence-control layer should show improved system-level properties resulting from energy awareness". System-level properties, according to Section 3.1, "are properties at the system level, i.e. of the full control system (shown in Figure 2.2) consisting of embedded control software, input/output hardware, and the plant". Therefore, the implementation of a sequence-control layer needs to be connected to a loop-control layer and plant (robot plus environment). Hence, a loop controller and robot are required to connect the sequence-control layer to. Since their development is out of this project's scope, they should be reused (Design Requirement 1) from another project.

Development of an energy-aware loop-control layer by RaM in the RobMoSys project (mentioned in Section 2.4) has not yet fully finished, so that specific loop controller is unavailable. Recently, Lazar (2019) developed a loop-control layer for haptic telemanipulation of a robotic arm. The setup comprises a real-world master robot-arm in open space and a slave robot-arm with a virtual wall in a simulation environment. Because this robotic setup can be used in future work of RaM and the code and documentation are readily available, it is taken as a starting point for the project treated in this thesis.

Ideally, a physical real-world robot and environment would be implemented. However, due to current government restrictions intended to limit the spread of COVID-19[4], access to physical robots at the university's labs is denied. Therefore, the robot is implemented in simulation. Hence, the robot-arm simulation and according slave loop controller by Lazar (2019) are used in this project. By using a robot-arm simulation, Implementation Requirements 4 and 5 are satisfied.

Because the loop controller was implemented in C++ and connected to the robot with ROS[5] middleware, and the robot was simulated in Gazebo[6], the same software is used in this project. Hence, the sequence-control layer is written in C++. Note that ROS does not provide real-time guarantees, so Implementation Requirement 1 is not satisfied. This should, however, not result in problems, because the loop controller (which has more strict real-time requirements than sequence control) and robot already proved to form a working system with ROS in the project of Lazar (2019). On the plus side, ROS is a widely used and supported framework in robotic applications, in line with Design Requirement 1.

[4] World Health Organisation web page on COVID-19: *https://www.who.int/health-topics/coronavirus/*
[5] ROS website: *https://www.ros.org/*
[6] Gazebo website: *http://gazebosim.org/*

The robot consists of a simulated KUKA LWR 4+ robotic arm in an open space simulation environment including a virtual wall. This corresponds to fault scenario I, Unexpected Obstacle, presented in Figure 3.6. As this scenario is readily available, it is reused in this project to satisfy Design Requirement 5a. The robot itself stands on a base and a side view of the setup is shown in Figure 4.8.

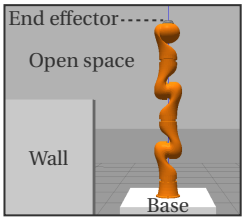


Figure 4.8: Side view of the simulated KUKA LWR 4+ robotic arm and its environment.

The loop controller controls the robot through a virtual spring attached between the setpoint and the robot's end-effector as illustrated in Figure D.1. The implemented loop controller can output an end-effector wrench vector or a joint-forces vector, either way corresponding to a generalised force (effort, $\boldsymbol{e}$) type output. As input, the loop controller takes an H-matrix that describes the setpoint (sp) for the robot end-effector pose (position plus orientation) with respect to the robot's inertial reference frame, $\boldsymbol{H}^0_{\text{sp}}$. The implication is that the sequence-control implementation should output this H-matrix to the loop controller. The loop controller used in this project does not contain energy guards or passivity layers and is not energy-aware. This is kept as is, because loop-control development is not in the scope of this project. Passivity and energy awareness are thus solely left up to the sequence controller.

In order to integrate energy awareness, the loop controller has been adapted to send potential- and kinetic energy to the sequence controller. The potential energy corresponds to the elastic energy contained by the virtual spring. The kinetic energy corresponds to the robot's kinetic energy. Diverting from the top-level design shown in Figure 4.2, but conform to the interface design presented in Section 4.1.1, these energies are sent directly from the loop-control layer to the sequence-control layer. This is done, because an energy supervisor—which should perform the task of energy transmissions and distributions between control layers—has not yet been developed. Hence, the implementation in this project does not contain, but allows adaptation to, an energy supervisor (Design Requirement 8a).

4.2.2 Interfaces

At the basis of the energy-aware sequence control implementation stand the control interface and the energy interface presented in Sections 3.2 and 4.1.1. They have been implemented as abstract classes and are used to communicate signals between components. A UML class diagram representation of how these interfaces have been implemented is shown in Figure 4.9. The 'Sequence Controller' top class in Figure 4.9 has been implemented to cover the 5Cs-concern of coordination (Design Requirement 1) over the sequence-control components, i.e. coordinate their life cycles and execution sequences. In accordance with the sequence-control layer's detailed design shown in Figure 4.3, each sequence-control component establishes both the control interface and the energy interface. Information on the private (-) port variables and the protected (#) and public (+) port functions shown in Figure 4.9 is given in Section D.2.

All getter functions have been applied such that they retrieve a port variable's value and then reset that value to zero (except for H-matrices, which are reset to identity), like leaving an empty mailbox after getting the mail. This is a safety feature for communication losses and prevents a port variable's value from repeatedly being read while not being updated. The interface warns the user when a component reads an empty port variable.

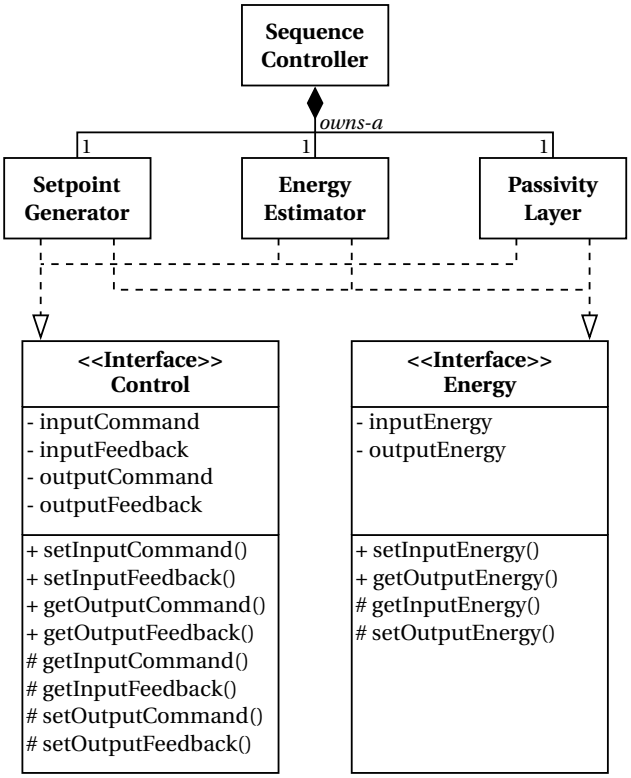


Figure 4.9: UML class diagram describing the implemented interfaces. Port variables are private (-) and functions are either public (+) or protected (#).

Control Interface

Figure 4.10 illustrates how the implemented control interface functions by showing an example component that incorporates the control interface. An explanation on the interpretation of Figure 4.10 through a practical example is given in Section D.2.1.

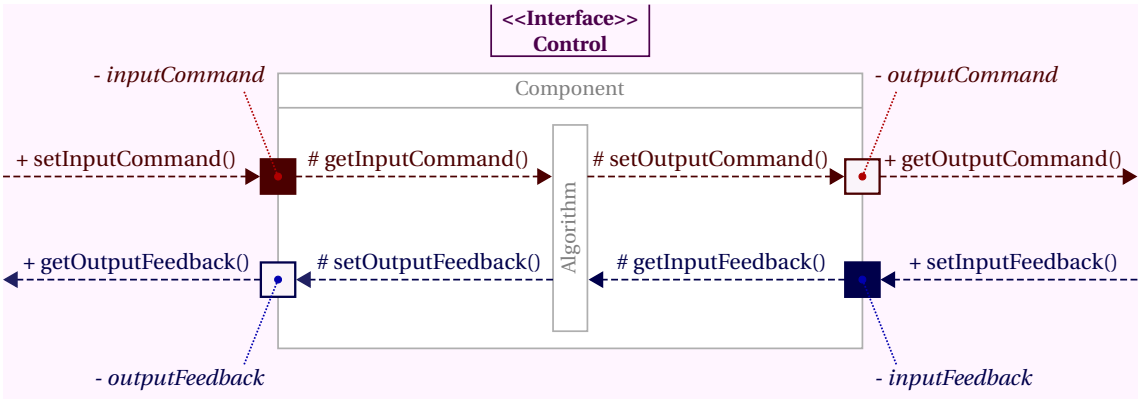


Figure 4.10: Illustration of the functionalities that the control interface provides to a component. The interface provides pre-defined functions to communicate any of the four variables of state (which can contain any of the quantities listed in Tables A.1 and A.2) between components. It provides 4 public (+) port functions, 4 protected (#) port functions, and 4 private (-) port variables.

For enabling signals to contain the four physical variables ($\boldsymbol{e}$, $\boldsymbol{p}$, $\boldsymbol{f}$, and $\boldsymbol{q}$), a signal type called `ControlSignal` has been defined as shown in Listing 4.1. The subtypes `eType`, `pType`, `fType`, and `qType`—that `ControlSignal` consists of—are defined in Listing D.1 and provide specific vector and matrix types for the physical quantities listed in Tables A.1 and A.2. This is to facilitate a physics-conformal connection of components. The port variables

shown in Figure 4.10 are of type `ControlSignal`, the get-functions return a signal of type `ControlSignal`, and the set-function only accept an argument that is of type `ControlSignal`.

Listing 4.1: Definition of the control-signal type belonging to the control interface written in C++ . The control-signal type is based on the four physical variable type definitions in Listing D.1.

```
1 // Control signal comprising the four physical variables
2 struct ControlSignal {
3     eType e;
4     pType p;
5     fType f;
6     qType q;
7 };
```

In line with the interface design explanation in Section 4.1.1, the control interface allows for the use of any and multiple of the physical variables (***e***, ***p***, ***f***, and ***q***) without validity checks. Therefore, a developer requires basic understanding of the physics concepts explained in Section 2.3 to make correct and physics-conformal use of the control interface.

Energy Interface

The principles of private port variables with public and protected port getters and setters, that have been described above for the control interface, also apply to the energy interface illustrated in Figure 4.11.

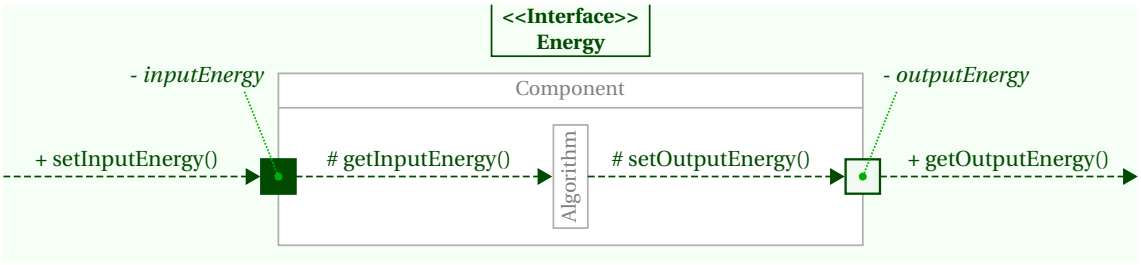


Figure 4.11: Illustration of the functionalities that the energy interface provides to a component. The interface provides pre-defined functions to communicate the seven energy variables show in Table 3.2 between components. It provides 2 public (+) port functions, 2 protected (#) port functions, and 2 private (-) port variables.

A main difference between the control interface and the energy interface is the type of signal being communicated. Whereas the control interface allows the transmission of physical quantities in the form of vectors/matrices, the energy interface interface allows the transmission of multiple scalar energy values as shown in Listing 4.2.

Listing 4.2: Definition of the energy-signal type belonging to the energy interface written in C++. The energy-signal type contains the energies listed in Table 3.2.

```
// Energy signal comprising 7 energy variables
struct EnergySignal {
    double Etransfer{ 0.0 };
    double Etank{ 0.0 };
    double Ekinetic{ 0.0 };
    double Epotential{ 0.0 };
    double Eelastic{ 0.0 };
    double Egravtitational{ 0.0 };
    double Etotal{ 0.0 };
};
```

4.2.3 Top-level Implementation of the Sequence-control Layer

Using the interfaces described in Section 4.2.2, the signals shown in Figure 4.12 have been implemented. These signals are in line with the necessary/unnecessary signal explanations given in Section 4.1. Since the implementation serves only as a proof of concept, the components have been implemented in a basic form and many of the unnecessary signals have not been included. Regarding unnecessary signals that have been included, the *task* signal has been included to satisfy Design Requirement 2 and the $robot_{\text{state}}$ signal has been included to support a specific energy estimator implementation (explained in Section 4.2.6).

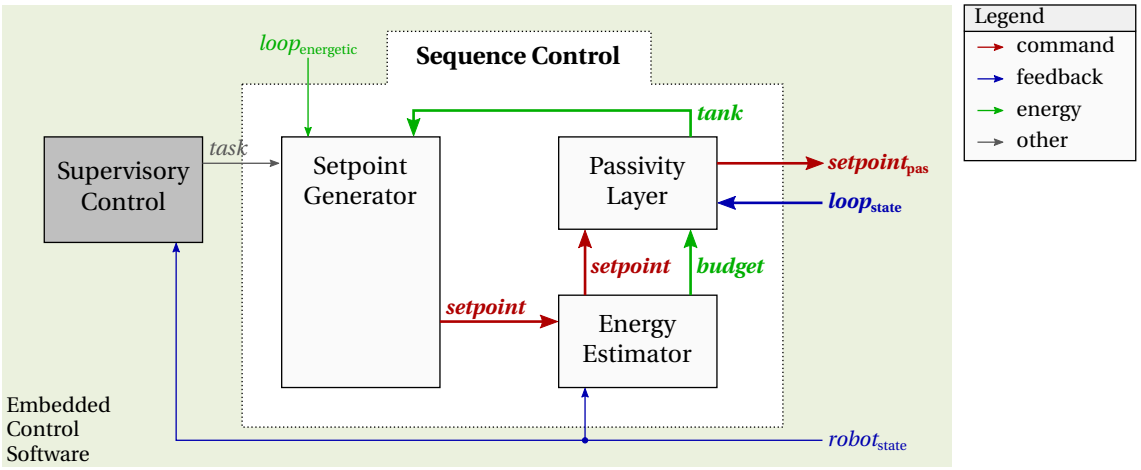


Figure 4.12: Implemented signals of Figure 4.3. Signal *task* contains a target H-matrix ($\boldsymbol{H}^0_{\text{target}}$) and signal *setpoint* contains a setpoint H-matrix ($\boldsymbol{H}^0_{\text{sp}}$).

In Section 4.2.1, it is stated that "as input, the loop controller takes an H-matrix that describes the setpoint (sp) for the robot end-effector pose (position plus orientation) with respect to the robot's inertial reference frame, $\boldsymbol{H}^0_{\text{sp}}$". The implication of this is that signal $setpoint_{\text{pas}}$ needs to contain this H-matrix. Therefore, the setpoints produced in the sequence-control layer have been implemented in the form of H-matrices (i.e. generalised displacements that fall under `qType` in Listing 4.1).

To illustrate the implemented transmission sequence of the signals in Figure 4.12, a sequence diagram is shown in Figure 4.13. The sequence of interactions loops at a fixed frequency. Note that the control sequence needs to loop, because the signals are interdependent, i.e. the command and feedback signals used in one control iteration are needed to calculate those in the next iteration.

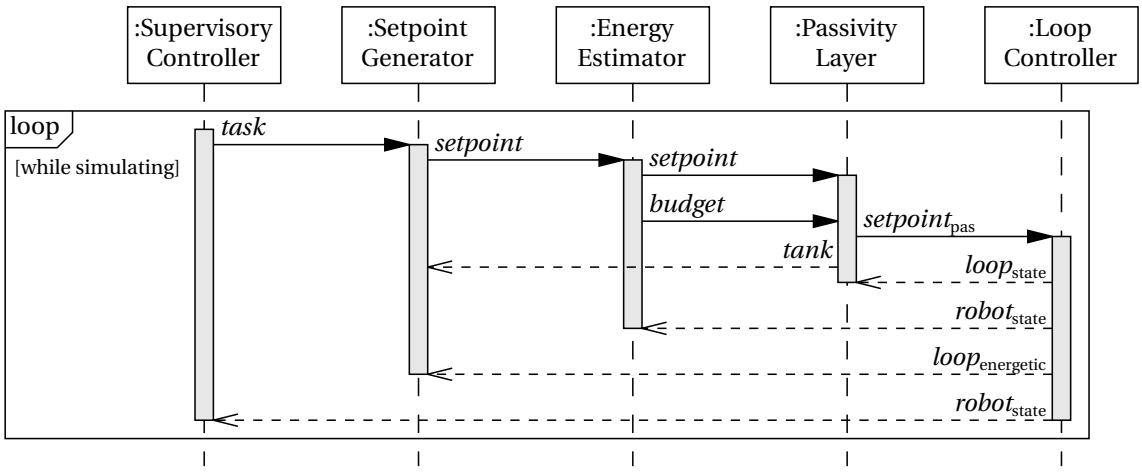


Figure 4.13: Sequence diagram showing the order in which the implemented interactions of Figure 4.12 take place during each control cycle. The cycle loops at a fixed frequency for as long the simulation is active.

Below follow explanations on the specific implementations of each of the components; supervisory controller (Section 4.2.4), setpoint generator (Section 4.2.5), energy estimator (Section 4.2.6), and passivity layer (Section 4.2.7).

4.2.4 Supervisory Controller

As can be seen from Figures 4.12 and 4.13, the implementation also includes a supervisory controller that writes a command signal, *task*, to the setpoint generator and receives a feedback signal, $robot_{\text{state}}$, from the loop controller. In line with the implementation of the sequence-control layer, the supervisory controller's output signal, *task*, has also been implemented to contain an H-matrix. This H-matrix describes a target pose of the robot's end effector with respect to the robot's inertial reference frame ($\boldsymbol{H}^0_{\text{target}}$). In turn, the setpoint generator's output signal, *setpoint*, contains a setpoint (sp) H-matrix, $\boldsymbol{H}^0_{\text{sp}}$, placed on a trajectory towards $\boldsymbol{H}^0_{\text{target}}$. An example configuration of body-fixed and commanded frames is given in Figure 4.14.

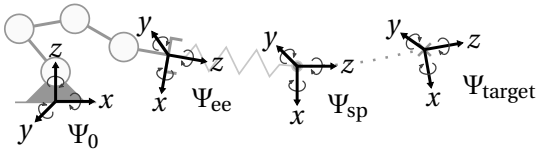
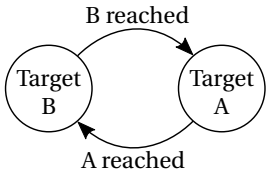
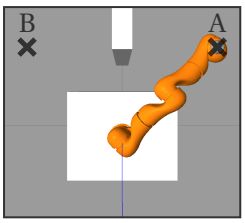


Figure 4.14: Extension of Figure D.1: illustrative placement of the inertial origin frame Ψ_0, robot end-effector frame Ψ_{ee}, setpoint frame Ψ_{sp}, and the target frame Ψ_{target}. A virtual spring modelled by the loop controller connects the robot end effector to the setpoint commanded by the sequence controller. The setpoint is placed on a trajectory towards the target commanded by the supervisory controller.

The supervisory controller has been implemented in line with Design Requirement 2. Because development of a supervisory controller has no priority in this project, it has been implemented as a simple state machine that switches its output between two task targets, as shown in Figure 4.15a. It reads and assesses the $robot_{\text{state}}$ signal to see if the robot has reached a target before switching to the other target. The two targets are placed on either side of the virtual wall to obtain fault scenario I: Unexpected Obstacle on the path between the two targets, as shown in Figure 4.15b. For ease of implementation and demonstration, the targets only changes position along the x-axis. Orientation is kept unchanged. Numerical details are given in Section D.3.



(a) State machine describing the supervisory controller's output state.



(b) Simulation top view of targets A and B outputted by the supervisory controller.

Figure 4.15: The supervisory controller's output switches between targets A and B. When the robot reaches a target, the supervisory controller switches its output from one target to the other.

4.2.5 Setpoint Generator

The main operations performed in each sequence-control component are presented in flowcharts. The flowcharts are looped together to indicate that execution of the sequence-control layer is repetitive. Connectors are used to link individual flowcharts to one another. Note: the flowcharts and component descriptions that follow below assume knowledge of design choices that are explained in Section 4.1.

The first flowchart is shown in Figure 4.16 and represents the start of the simulation. It serves as an initialisation and is executed only once. A boolean called *obstacle* (used in Figure 4.17) is initialised to *false* and a boolean called *budgetGen* (used in Figure 4.19) is read as input from the user. *budgetGen* stands for simulation type 'budget generation' (explained in Section 4.2.6) and is the only available user input, covering the 5Cs-concern of Configuration (Design Requirement 1).

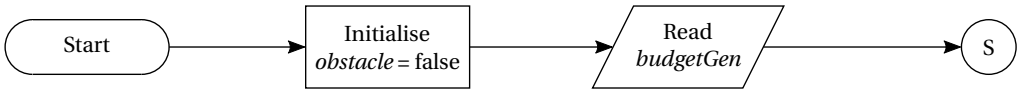


Figure 4.16: Flowchart 1/5: start of the simulation. Connector 'S' links to the setpoint generator flowchart in Figure 4.17.

As indicated by the sequence diagram in Figure 4.13, the first active sequence-control component is the setpoint generator. The implementation of this component is in line with its design shown in Figure 4.5 and the implemented signals shown in Figure 4.12. The main functionalities of this component are illustrated by the flowchart in Figure 4.17.

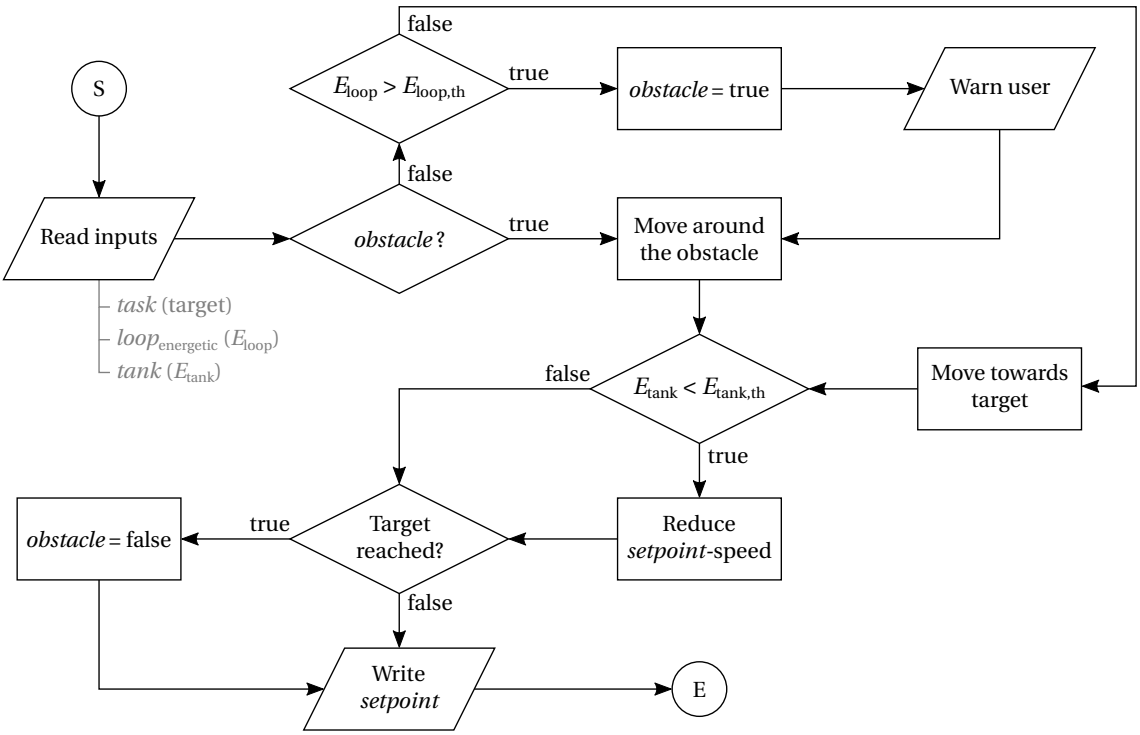


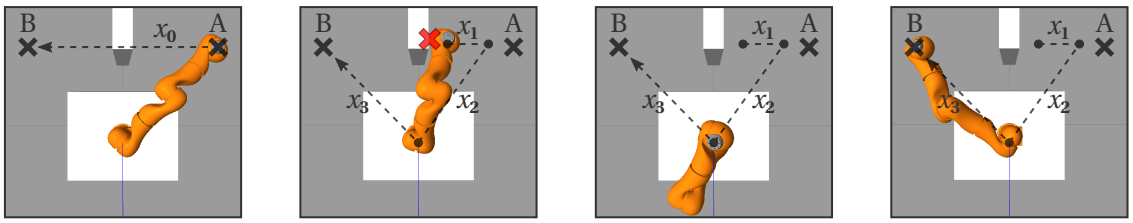
Figure 4.17: Flowchart 2/5: setpoint generator main functionalities. Connector 'E' links to the energy estimator flowchart in Figure 4.19.

The implemented setpoint generator outputs setpoints that move towards the received target in a straight line trajectory. Because fault scenario I: Unexpected Obstacle is implemented (Section 4.2.1, Implementation Requirement 6a), the setpoint generator has been implemented with obstacle detection functionality. Obstacle detection is performed by comparing the loop-control feedback energies to a list of conditions that are encapsulated by the '$E_{loop} > E_{loop,th}$' statement in Figure 4.17 (where 'th' stands for threshold). If the setpoint generator detects an obstacle, it sends a warning to the user and performs an alternative trajectory in an attempt to move around the obstacle. This process is illustrated in Figure 4.18. The implemented alternative trajectory consists of three stages:

x_1. Moving backwards 2/3rd of the way towards the starting position of the original trajectory while maintaining an amplitude-continuous signal (Design Requirement 5). Since the robot has already traversed this path prior to the collision, it should be possible to move back the same way. 2/3rd of the way back is taken as an estimate, assuming that the robot has cleared the obstacle with sufficient distance to be moving around again.

x_2. Moving towards the base of the robot, to 2/3rd of the initial configuration height. Since the robot is mounted to a base, it is expected that the robot can move towards it. Of course, there are situations imaginable where there is another obstacle in between, but this project's implementation only serves as a simple proof of concept. 2/3rd of the height is taken, because completely moving to the initial configuration—in which the robot arm stretches out straight up—would unnecessarily consume more energy.

x_3. Attempting to reach the target again by moving from the position at the base towards the target.



(a) t_0: planned straight line trajectory from A to B (x_0).

(b) t_1: collision detection and correction planning (x_1, x_2, x_3).

(c) t_2: traversing the alternative trajectory, i.e. x_1, x_2 and x_3.

(d) t_3: target reached.

Figure 4.18: Top view of the implemented collision correction approach. When a collision is detected the following correction is attempted: x_1) move 2/3rd of the way back to A, x_2) move to the base, x_3) move to the target (B).

Upon reaching any trajectory (sub)target, the setpoint generator briefly waits before moving its setpoints towards the next target. The reason for this is to have the robot—which lags behind the setpoint—more closely reach the (sub)target. Since the supervisory controller's commanded targets only change positions, the setpoint generator in this project has also been implemented for translational setpoints. It uses the xyz-coordinates of a commanded H-matrix and disregards rotations.

Under normal circumstances, the setpoint generator outputs its H-matrix setpoints at a preset speed (which, given a fixed loop frequency, equates to a pre-set distance between subsequent setpoints). However, when the passivity layer's energy-tank level gets below a certain threshold, the setpoint generator lowers the setpoint speed proportionally to the energy-tank level (the lower the energy-tank level, the lower the setpoint speed). This is done, because from Equation A.3 it can be seen that a lower setpoint speed (corresponding to a lower velocity/flow) reduces the amount of energy extracted from the sequence controller, thereby counteracting full depletion of the energy tank. Note that this is a redundant measure if the energy estimator provides fully accurate energy budgets to the passivity layer's energy tank.

A safety feature implemented in the setpoint generator is that it sets the setpoint speed to zero if it detects a communication loss with the loop controller. A communication loss is assumed if all of loop control's feedback signals are exactly equal to those during the previous control iteration. Note that safety features such as this one need to be included by the software developer as they are not enforced by the control architecture itself.

4.2.6 Energy Estimator

From the sequence diagram in Figure 4.13, it can be seen that the energy estimator is active after the setpoint generator. Even though Brodskiy (2014) presented an energy estimator design, no readily available energy estimator implementation for the given robot has been found. Therefore, a new energy estimator implementation needed to be developed. The energy estimator component has been implemented in line with the implemented signals shown in Figure 4.12 and in line with—but with an alternative take on—its design shown in Figure 4.6.

In Section 4.1.5 it is stated that "when implementing an energy estimator, during any trade-off involving accuracy, the importance of accurate budget estimations (and, hence, accurate models) for system performance should be kept in mind (Design Requirement 6b)". However, the implementation of this project serves as a basic proof of concept for the overall sequence-control layer. Therefore, it is not in the scope of this project to develop highly-accurate component models for the energy estimator. Nevertheless, energy-budget estimations of sufficient accuracy are needed to mitigate the negative effect of inaccurate energy-budgets on system performance, i.e. to not compromise Implementation Requirement 3.

One alternative possibility would be to run a reference simulation parallel to the normal simulation, i.e. a 'perfect world' reference simulation—without virtual wall—for the energy estimator to extract energy data from. This approach has been attempted, but implementing two parallel simulations in Gazebo that exchange information with each other through ROS was found to be a challenging task infeasible within the scope of this project. Instead, a solution that makes use of two serial simulations has been implemented (execution instructions are given in Appendix E):

1. **The budget-generation simulation** establishes energy budgets that are needed during the normal simulation. The budget-generation simulation comprises a perfect-world reference simulation (i.e. without virtual wall) with unlimited available energy. Each control iteration, the amount of energy extracted from the passivity layer by the loop controller is written on a new row in a log file, $\log_{\mathrm{E}}$. The logged energy values serve as energy budgets in the normal simulation. The H-matrix describing the robot's end-effector pose with respect to the inertial frame (contained by the $robot_{\mathrm{state}}$ signal) is also written to a log file (in CSV-format), $\log_{\mathrm{H}}$. The logged H-matrices are used for comparison to the robot's H-matrix during the normal simulation.
2. **The normal simulation** represents the realistic non-perfect world and can include the virtual wall. It runs the 'normal' scenario in which the energy estimator provides energy budgets to the passivity layer that it reads from $\log_{\mathrm{E}}$. To determine which specific energy budget to extract from $\log_{\mathrm{E}}$, the robot's real pose (H-matrix) is compared to the H-matrices in $\log_{\mathrm{H}}$. The row index of the best-matching H-matrix is then taken as the same row index in $\log_{\mathrm{E}}$ to read the energy budget from.

H-matrix matching is used to determine the most suitable energy budget during the normal simulation, because this approach allows the provision of energy budgets based on the actual state of the robot. This is a dynamic approach, unlike energy-budget provisions based on simulation time would be. If the setpoint generator would decide to output an alternative trajectory, energy estimation based on H-matrix matching will continue functioning (albeit less accurately). More detailed explanations on the implemented H-matrix matching is given in Section D.4.

The serial simulation approach is an alternative take on the energy estimator design in Section 4.1.5 in the sense that it does not strictly consist of a Model of Loop Control, Model of Robot, Model of Environment, and Energy Sampling blocks. It can be seen as running an energy-sampled simulation consisting of those model blocks and storing the results in a lookup table so that the lookup table itself represents the design in Section 4.1.5. This approach is sufficient here, because it serves the project's proof-of-concept purpose while complying to the interfaces in the main architecture design of Figure 4.3. Nevertheless, future work should more closely follow the energy-estimator design shown in Figure 4.6 and actually implement computational component models.

The main functionalities of the energy estimator implemented in this project are illustrated by the flowchart in Figure 4.19.

The variable $E_{\mathrm{estimated}}$ in the flowchart has been implemented as follows:

$$E_{\mathrm{estimated}} = \frac{E_{\mathrm{row}} + E_{\mathrm{average}}}{2}, \tag{4.3}$$

in which E_{row} represents the energy value extracted from the best-matching row in $\log_{\mathrm{E}}$ and E_{average} represents the average over all energies in $\log_{\mathrm{E}}$. This equation effectively adds a constant component (E_{average}) to the estimated energy budget without altering the total energy-budget provision over a complete simulation (i.e. without compromising passivity). This has been implemented, because it was empirically found to result in less passivity-layer triggers for

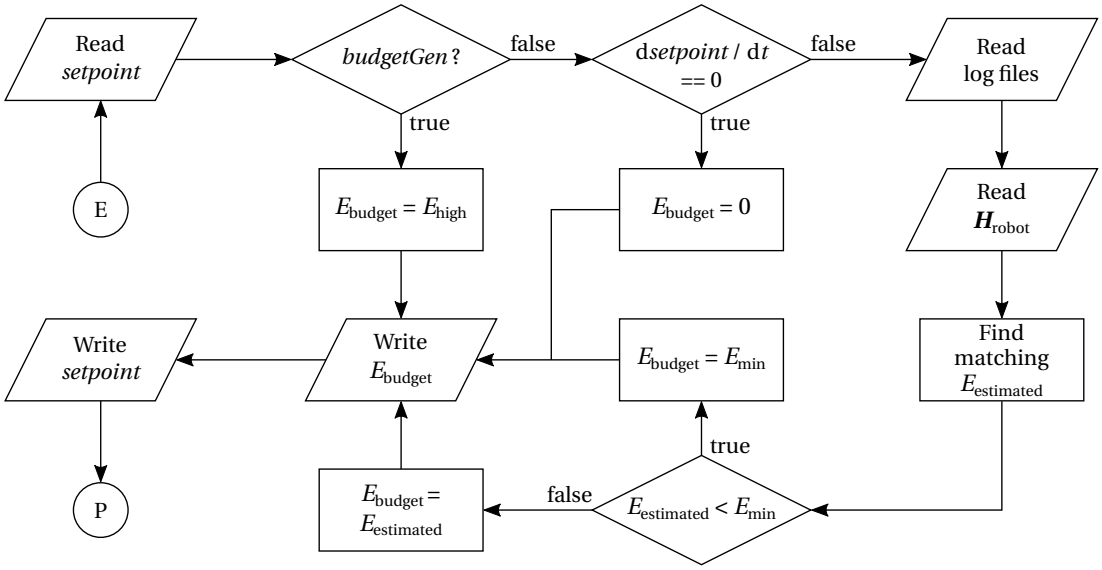


Figure 4.19: Flowchart 3/5: energy estimator main operations. Connector 'P' links to the passivity layer flowchart in Figure 4.20.

multiple types of trajectories. $E_{average}$ corresponds to the average energy consumed by the loop controller during a complete budget-generation simulation. Assuming that, during a budget-generation simulation, the subsystem comprising of the loop controller, robot, and environment is passive (passive systems, by definition, do not generate energy), $E_{average}$, by definition, has a positive value.

As indicated by the flowchart: if $E_{estimated} < E_{min}$, then $E_{budget} = E_{min}$. Hence, a minimum energy budget, E_{min}, has been implemented. For this project, it has been empirically set to a relatively low value of 0.01 J. The purpose of E_{min} is twofold:

1) It prevents the energy budget from becoming negative so that the energy estimator cannot extract energy from the passivity layer (the energy estimator's purpose is to provide energy, not consume it). The reason that energy values in $\log_E$ can be negative is that the robot can return energy into the controller when forces and velocities are in opposite direction. This can occur during deceleration, or during counteraction of movements imposed by an active environment.

2) It prevents the energy budget from being equal to zero so that situations of stagnation cannot occur. The reason that stagnation may occur when energy budgets of zero Joule are allowed, is that it then becomes possible to have situations in which the energy tank runs out of energy, the robot comes to a halt, and energy budgets of zero Joule are provided. The result is then effectively a deadlock in which the robot cannot move due to an empty energy tank, the energy tank waits for new energy budgets, and the energy budgets are zero for as long as that position is maintained. Forcing a low, but non-zero, energy budget removes the possibility of that deadlock.

The flowchart also indicates that the energy budget is set to zero if the setpoint velocity equals zero: $E_{budget} = 0$ if $v_{setpoint} == 0$. This is because the loop controller's energy consumption, by definition (Equation A.3), equals zero when the setpoint velocity is zero. Hence, no energy budget should be needed in that case. This situation can, for example, occur if the setpoint generator incorporates moments of zero setpoint velocity during the performance of an alternative (correction) trajectory, as is the case in this project.

4.2.7 Passivity Layer

The sequence diagram of Figure 4.13 shows that the passivity layer is the last active sequence-control component. The passivity layer has been implemented in line with its design shown in Figure 4.7 and the implemented signals shown in Figure 4.12. Like its design, the passivity layer's core functionality implementation has been inspired from available passivity layers within the RaM research group (in line with Design Requirement 6ai). The main functionalities of this component are illustrated by the flowchart in Figure 4.20.

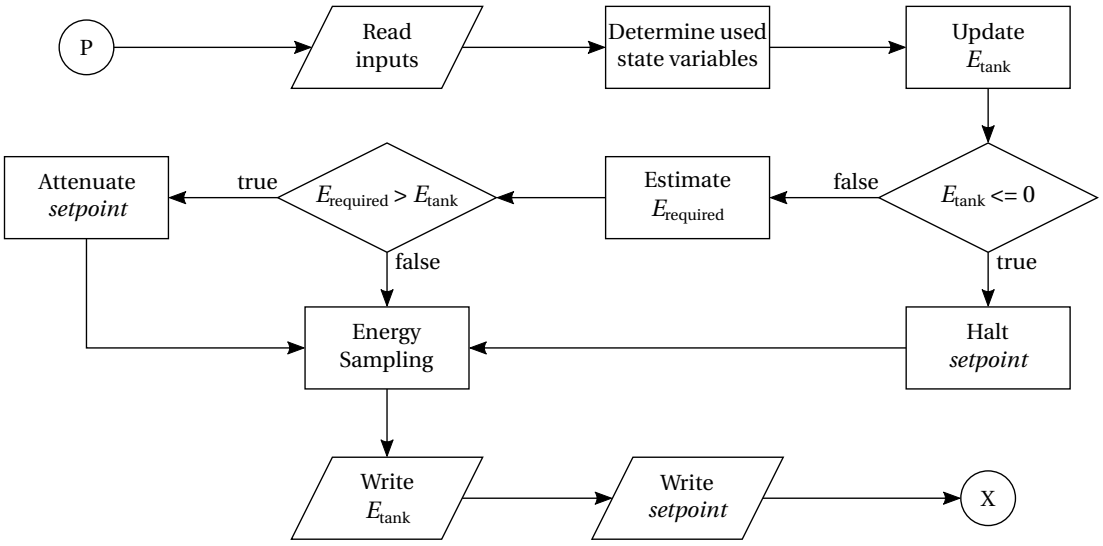


Figure 4.20: Flowchart 4/5: passivity layer main operations. Connector 'X' links to the simulation end in Figure 4.21.

The passivity layer has been made compatible to the control- and energy interfaces described in Sections 4.1.1 and 4.2.2. In line with the design presented in Section 4.1.6, Equation 3.5 is used to calculate energy exchanges and the setpoint is passively modulated following Equation 4.2. The estimated required energy exchange in Equations 4.1 and B.4 is calculated via estimation of the 4 physical variables ($\boldsymbol{e}^*_{k+1}$, $\boldsymbol{p}^*_{k+1}$, $\boldsymbol{f}^*_{k+1}$, or $\boldsymbol{q}^*_{k+1}$). Estimation of these variables is implemented as a first-order approximation, i.e. assuming that the rate of change at sample time t_k is equal at sample time t_{k+1}:

$$\begin{aligned} \boldsymbol{x}^*_{k+1} &= \boldsymbol{x}_k + \frac{\mathrm{d}\boldsymbol{x}_k}{\mathrm{d}t_k} \cdot \mathrm{d}t^*_{k+1} \\ &= \boldsymbol{x}_k + \frac{\boldsymbol{x}_k - \boldsymbol{x}_{k-1}}{t_k - t_{k-1}} \cdot \mathrm{d}t^*_{k+1}. \end{aligned} \tag{4.4}$$

$\mathrm{d}t^*_{k+1} = t^*_{k+1} - t_k$ represents the expected time difference between time samples k and $k+1$, i.e. the period of the control frequency. $\boldsymbol{x}$ is used as a generic placeholder for any variable.

The passivity layer is implemented such that it accepts any of the combinations listed in Table 3.1 in the form of variables from the mechanical domain listed in Table A.1 and from screw theory listed in Table A.2. If multiple of such variables are provided, the implemented passivity layer autonomously decides which ones to make use of. It sends warnings to the user if incompatible combinations are used, for example an H-matrix command signal together with a twist-feedback signal. This feature helps ensuring physics-conformity of the sequence-control layer. Unlike the setpoint generator and energy estimator (which only function with H-matrices), the implemented passivity layer is implementation-agnostic. It can be inserted in any sequence-control system that satisfies the design requirements of Section 3.5.1.

Figure 4.21 shows the flowchart that marks the end of a simulation iteration. This flowchart connects back to the setpoint generator flowchart (Figure 4.17) for as long as the simulation runs. The simulation terminates when the user commands it to.

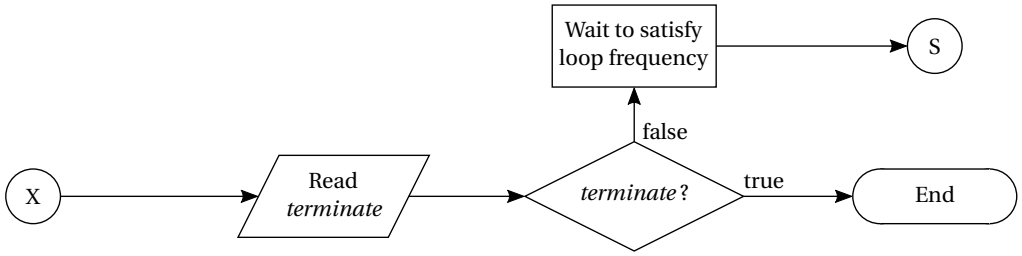


Figure 4.21: Flowchart 5/5: end of the simulation. Connector 'S' links to the setpoint generator flowchart in Figure 4.17.

4.3 Conclusion

This chapter describes the design of an energy-aware sequence-control layer and its conceptual implementation on the basis of the design and implementation requirements posed in Section 3.5. The design serves as a architectural guide for the implementation of energy-aware sequence control components. The implementation described in this chapter is conceptual and serves a proof of concept. It is a take on how the design could be implemented. The sequence-control layer design and implementation consist of two interfaces and three components: the control interface, the energy interface, the setpoint generator, the energy estimator, and the passivity layer. The combination of these defines the full energy-aware sequence-control layer's functionality.

The setpoint generator outputs amplitude-continuous setpoint commands based on input tasks from the supervisory-control layer and energy feedback. The energy estimator provides energy budgets to the passivity layer's energy tank. The passivity layer enforces a passive connection between the sequence-control layer and the loop-control layer. The control signals and energy signals are communicated using the control interface and the energy interface, respectively. The system has been implemented in simulation and uses a virtual wall to achieve Fault scenario I: Unexpected Obstacle (presented in Section 3.4).

The design requirements of Section 3.5.1 have all been satisfied. This does not hold true for the implementation requirements of Section 3.5.2 as Implementation Requirement 1 has not been satisfied due to ROS being non real-time. Yet, this should not result in problems, because the loop controller (which has more strict real-time requirements than sequence control) and robot already proved to form a working system with ROS in the project of Lazar (2019). Implementation Requirements 4, 5, and 6, on the other hand, have been satisfied and the remaining implementation requirements, Implementation Requirements 2 and 3, are treated in Chapter 5. Despite that not all implementation requirements are satisfied in this chapter, it does indicate that the project's subgoal of "designing and implementing the architecture of an energy-aware sequence-control layer", stated in Section 1.3, has been fulfilled.

5 Evaluation

This chapter treats the project's subgoal of "demonstrating improved system-level properties through energy awareness", stated in Section 1.3. Four different experiments on non-energy-aware and energy-aware sequence control are treated. With the results of these experiments, the design and implementation of the energy-aware sequence-control architecture presented in Chapter 4 are evaluated on the basis of the system-level properties of safety, dependability, and performance, following Implementation Requirements 2, 3 and 6.

5.1 Experiments

The experiments that have been conducted in this project serve a common purpose of demonstrating system-level properties, such as energy awareness, of the sequence-control implementation (Implementation Requirements 3 and 6). Four experiments have been conducted, each with the same duration and the same x-position targets (targets A and B, explained in Section 4.2.4). The differences between the four experiments are in the form of simulation types (i.e. 'normal' and 'budget generation', explained in Section 4.2.6), and environmental and energy-budget conditions, as explained below:

1. **Using a single energy budget – without wall**
   In the first experiment, the passivity layer's energy tank starts at an initial level of 75 J. The simulation runs without a virtual wall and the energy estimator does not output energy budgets. 75 J has empirically been chosen as a starting level, because it leads to insightful data where the energy tank runs out of energy before reaching target B. The objective of this experiment is to evaluate the setpoint generator's functionality of energy tank-based velocity reduction (described in Section 4.2.5) and to verify system passivity.
   Experiment details:
   - Type: normal simulation
   - Energy budgets: 75 J provided only once, at the start of the simulation
   - Virtual wall: no

2. **Finding energy budgets – without wall**
   The second experiment performs the budget-generation simulation explained in Section 4.2.6. The unlimited available energy and absence of the virtual wall prevents energy-guided decisions from being made, i.e. this experiments equates to non-energy-aware sequence control. The simulation is used to establish the energy budgets that the energy estimator outputs during a normal simulation. The energy budgets are obtained from the amount of energy consumed by the loop controller at each control cycle. This experiment also serves as a reference scenario to compare with the other experiments.
   Experiment details:
   - Type: budget-generation simulation
   - Energy budgets: are measured and written to a log file
   - Virtual wall: no

3. **Using energy budgets – without wall**
   The third experiment performs a normal simulation, i.e. the energy estimator uses the energy budgets obtained during the second experiment. The use of energy budgets means that a limited amount of energy is available, which is the only difference between this experiment and the second experiment. The purpose of this experiment is to show the effect that enabling energy awareness has on system-level properties.
   Experiment details:
   - Type: normal simulation

- Energy budgets: read from the log file and outputted by the energy estimator
- Virtual wall: no

4. **Using energy budgets – with wall**
   The fourth and final simulation comprises a normal simulation that includes the virtual wall, corresponding to fault scenario I: Unexpected Obstacle. The purpose of this experiment is to fulfil Implementation Requirements 3 and 6 and demonstrate the sequence controller's energy awareness and other improved system-level properties.
   Experiment details:
   - Type: normal simulation
   - Energy budgets: read from the log file and outputted by the energy estimator
   - Virtual wall: yes

In all experiments, the setpoint generator's standard setpoint motion speed is set to 0.75 m/s. Experiments 2, 3 and 4 each run for a duration of 7.56 seconds (set equal to the time it takes to reach target B in experiment 4) to allow for easy comparison between the three experiments. In all experiments, the loop controller runs at a frequency of 1 kHz and the sequence controller at 100 Hz (performance limitations of the host computer do not allow a higher control frequency). A list of hardware and software materials that have been used in the experiments is given in Appendix F. The results of the experiments are presented in Section 5.2.

5.2 Results

This section presents the data resulting from the four experiments described in Section 5.1. For each experiment, a figure consisting of four subfigures is shown. The left side of each figure presents position data and the right side of each figure presents energy data. Subfigures (a) present xy-position data, that is, a top view of the setpoint and robot positions (contained by signals *setpoint* and $robot_{\text{state}}$, respectively, in Figure 4.12). Note that, thus, the presented setpoint data is of the setpoint generator's output—not the passivity layer's. Because the targets change only along the x-axis, subfigures (b) present the x-position data of subfigure (a) versus time. Subfigures (c) present energy exchange data versus time—consisting of provided and consumed energy levels, energy-tank levels, loop-control energy levels, and energy budgets. The total energy provided by the energy estimator to the passivity layer equals the sum of provided energy budgets (contained by signal *budget* in Figure 4.12),

$$E_{\text{provided},k} = \sum_{i=1}^{k} \Delta E_{\text{budget},i}\,, \tag{5.1}$$

and the total consumed energy equals the sum of energy exchanges between the loop controller and the passivity layer,

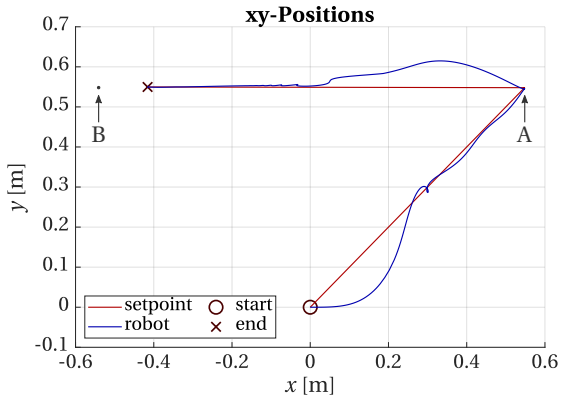
$$E_{\text{consumed},k} = \sum_{i=1}^{k} \Delta E_{\text{consumed},i}\,, \tag{5.2}$$

with $\Delta E_{\text{consumed},i}$ defined by Equation 3.5 and calculated from H-matrix commands and wrench feedbacks (contained by signals $setpoint_{\text{pas}}$ and $loop_{\text{state}}$, respectively, in Figure 4.12). The loop-control energy level shown in subfigures (c) equals the sum of kinetic and potential energy, both of which are contained by the signal $loop_{\text{energetic}}$ in Figure 4.12:

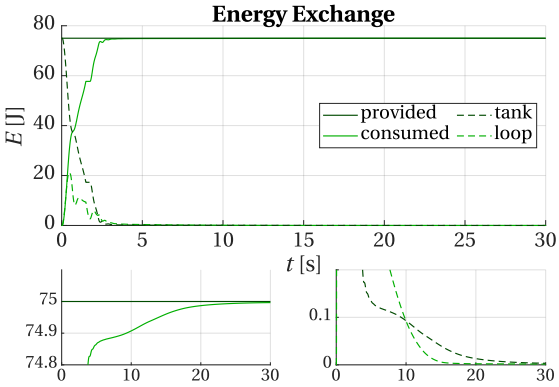
$$E_{\text{loop},k} = E_{\text{kinetic},k} + E_{\text{potential},k}\,. \tag{5.3}$$

Subfigures (d) present energy-state data consisting of the energy-tank level (also shown in subfigure (c)) and the potential- and kinetic energies from the loop-control layer (whose sum is calculated with Equation 5.3 and shown in subfigure (c)).

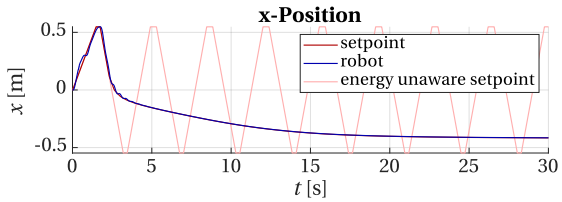
5.2.1 Experiment 1: Using a Single Energy Budget – Without Wall



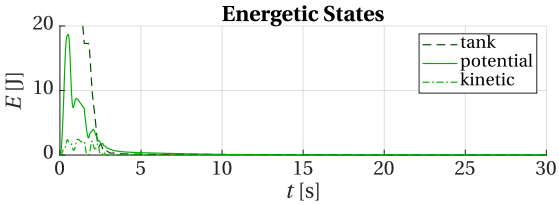
(a) Top view of the setpoint and robot end-effector positions. Both start at the robot's base. Targets A and B are set by the supervisory controller.



(c) Energy exchange time series. The bottom left and right plots zoom in on the top and bottom sections of the data, respectively. 'provided' represents the starting energy of 75 J. 'tank' represents the energy-tank level. 'consumed' represents the amount of energy extracted from the energy tank by the loop controller. 'loop' represents the sum of potential and kinetic energy available at the loop-control layer.



(b) Time series of the setpoint and robot end-effector x-position. 'energy unaware setpoint' represents what the setpoint would have been in case of non-energy-aware setpoint generation (such as in Figure 5.2b).

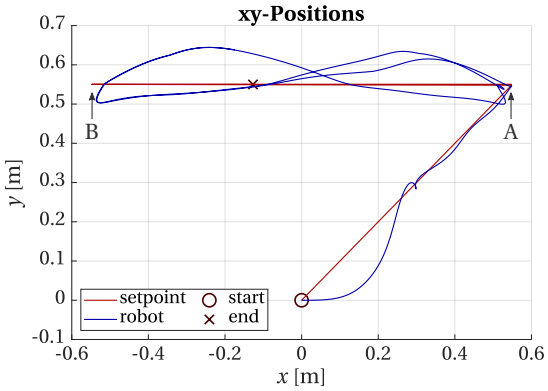


(d) System energetic states time series. 'tank' represents the energy-tank level (equal in Figure 5.1c). 'potential' and 'kinetic' represent the potential and kinetic energies available at the loop-control layer, respectively.

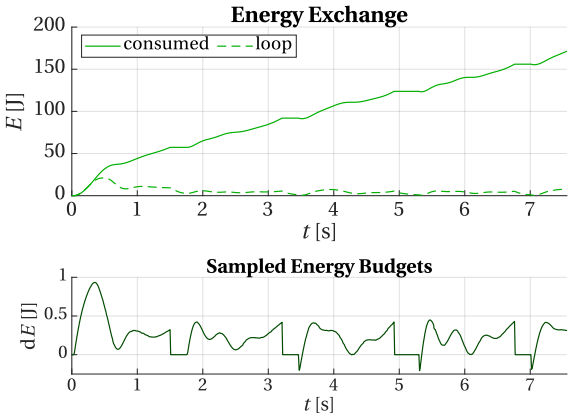
Figure 5.1: Position and energy data of the first experiment, which uses a starting energy-tank level of 75 J without further energy budgets and where the virtual wall is absent.

Figures 5.1a and 5.1b shows that the setpoint ends at $x = -0.4159$ m and the robot's end effector ends at $x = -0.4155$ m. Thus, the setpoint (and, hence, also the robot) does not reach target B. From Figure 5.1b it can be seen that this is due to the setpoint generator's output setpoint gradually approaching zero velocity. From Figure 5.1c it can be seen that the setpoint velocity reduction occurs concurrently with energy tank depletion. It can also be observed that the consumed energy approaches—but does not exceed—the 75 J that was provided to the energy tank. Figure 5.1d shows a decrease in all energetic states as time progresses.

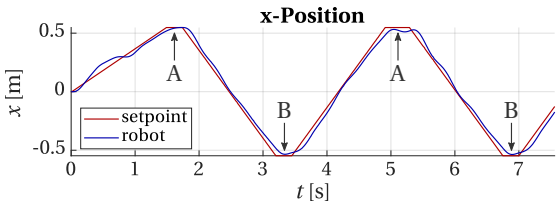
5.2.2 Experiment 2: Finding Energy Budgets – Without Wall



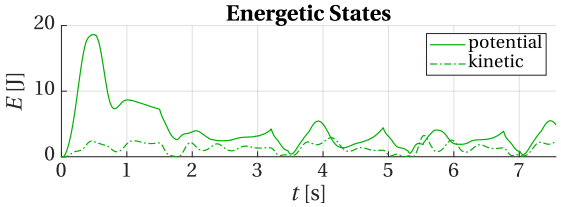
(a) Top view of the setpoint and robot end-effector positions.



(c) Energy exchange time series. The top graph shows the the total consumed energy alongside the loop controller's total energy (which equals the sum of the two energies shown in Figure 5.2d). The bottom graph shows the obtained energy budgets, i.e. the difference signal of 'consumed' in the top graph ($E_{\text{budget},k} = E_{\text{consumed},k} - E_{\text{consumed},k-1}$).



(b) Time series of the setpoint and robot end-effector x-position. Targets A and B correspond to those in Figure 5.2a. The setpoint corresponds to the energy unaware setpoint shown in Figure 5.1b.

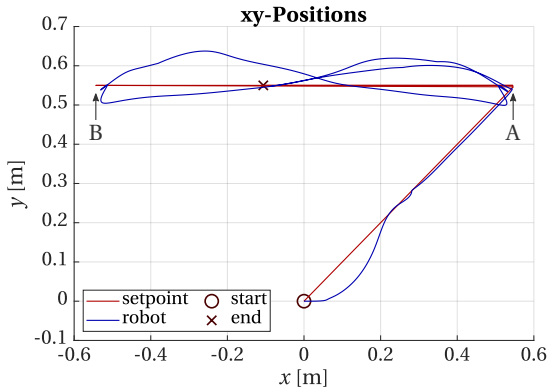


(d) Time series of the potential and kinetic energies available at the loop controller.

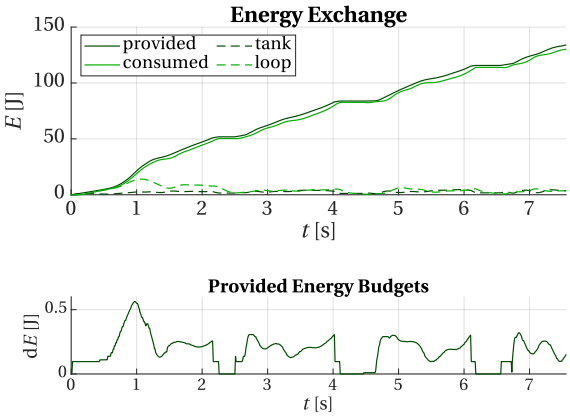
Figure 5.2: Position and energy data of the second experiment, which involves an energy-unbounded (equivalent to non-energy-aware) simulation without virtual wall and is used to obtain energy budgets for the energy estimator.

Figure 5.2a shows that the setpoint and robot move unobstructed between targets A and B, which can also be observed from Figure 5.2b. Figure 5.2b shows that the setpoint moves at a constant velocity of the specified 0.75 m/s between targets A and B, except for the starting movement from the origin to target A, which takes place at a lower x-velocity. Figure 5.2b also shows that the setpoint reaches targets A and B at $x = 0.548$ m and $x = -0.547$ m, respectively, i.e. not precisely at 0.55 m and -0.55 m specified in Section 4.2.3. Figure 5.2c shows that the loop controller consumes a total of 171.3 J over 7.56 seconds. It also shows that the sampled energy budget—immediately after a moment in which the setpoint velocity equals zero (e.g. at $t = 3.46$ s)—can be briefly negative. Figure 5.2d shows that the potential energy becomes relatively large at the start of the simulation, i.e. when the robot is being displaced from its inert starting configuration.

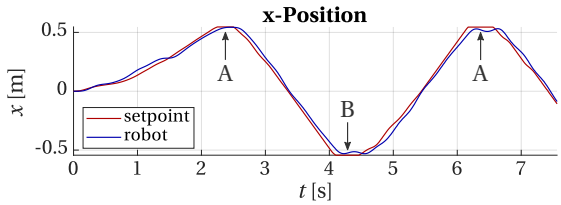
5.2.3 Experiment 3: Using Energy Budgets – Without Wall



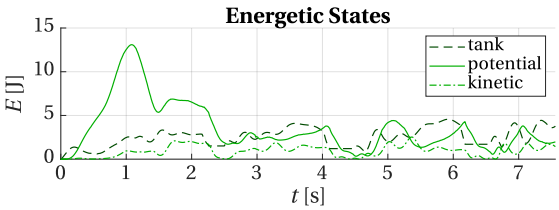
(a) Top view of the setpoint and robot end-effector positions.



(c) Energy exchange time series. The top graph shows the total energy provided by the energy estimator and the total energy consumed by the loop controller together with the energy-tank level and the loop controller's total energetic state. The bottom graph shows the energy budgets outputted by the energy estimator.



(b) Time series of the setpoint and robot end-effector x-position.

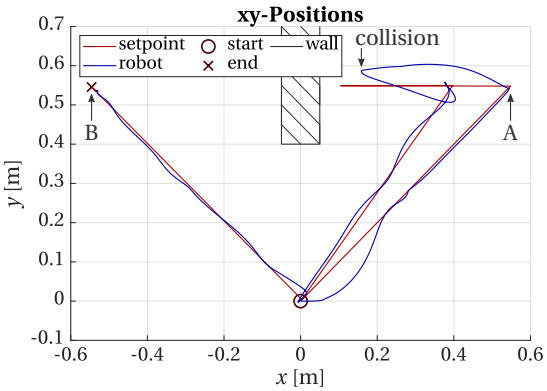


(d) System energetic states time series. 'tank' corresponds to its equal in Figure 5.3c.

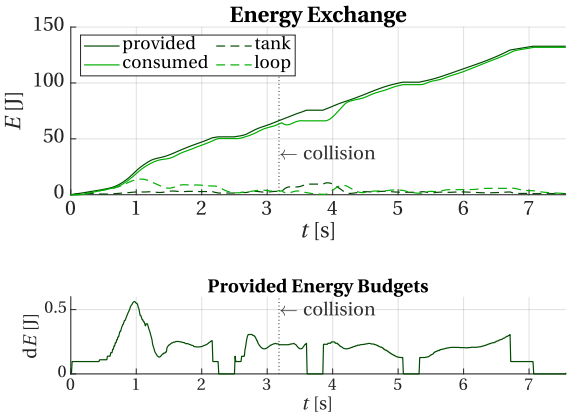
Figure 5.3: Position and energy data of the third experiment, which covers a normal simulation without virtual wall. This experiment is equivalent to the second experiment shown in Figure 5.2, except that in this experiment the energy estimator provides energy budgets that have been obtained during the second experiment.

Figure 5.3a shows that the setpoint and robot move unobstructed between targets A and B. It appears as if the simulation ends at approximately the same position as in the second experiment shown in Figure 5.2a. However, Figure 5.3b shows that the setpoint moves slower than in Figure 5.2b, particularly at the start of the simulation. It shows that the setpoint ends at $x = -0.1062$ m along a trajectory from target A to target B, equivalent to a delay of 1.4 seconds compared to the second experiment shown in Figure 5.2b. Figure 5.3c shows that a total of 134 J is provided by the energy estimator and a total of 130.3 J is consumed by the loop controller. It also shows that the provided energy-budget time series is similar in shape to the obtained energy budgets shown in Figure 5.2c, but with differences most notable at the start of the simulation. Figure 5.3d shows similar energetic states as in Figure 5.2d, but significantly lower at the start of the simulation.

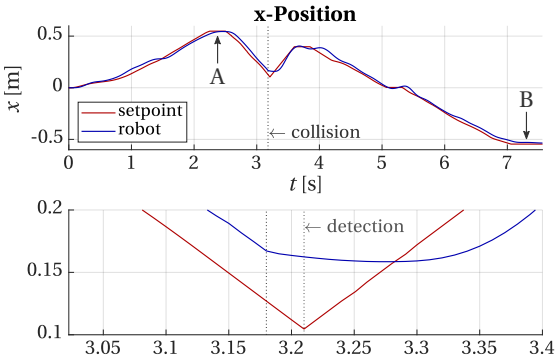
5.2.4 Experiment 4: Using Energy Budgets – With Wall



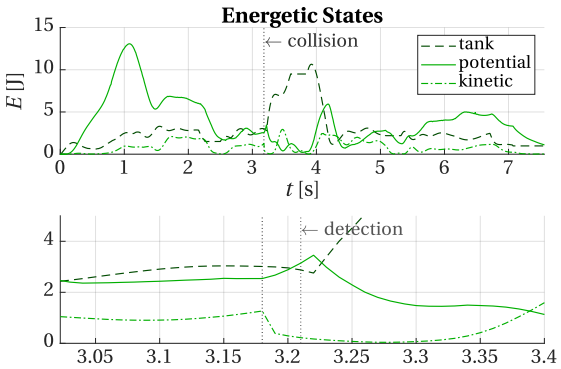
(a) Top view of the setpoint and robot end-effector positions. The robot end-effector position at which the robot collides with the wall is marked. The simulation ends after the setpoint reaches target B for the first time. This plot corresponds to the simulation screenshots shown in Figure 4.18.



(c) Energy exchange time series. The top graph shows the total energy provided by the energy estimator and consumed by the loop controller together with the energy-tank level and the loop controller's total energetic state. The bottom graph shows the energy budgets outputted by the energy estimator.



(b) Time series of the setpoint and robot end-effector x-position. The time instance of the wall collision is indicated. The bottom graph zooms in at that time instance and indicates the time at which the setpoint generator detects the collision.



(d) Time series of the system's energetic states. The time instance of the wall collision is indicated. The bottom graph zooms in at that time instance and indicates the time at which the setpoint generator detects the collision.

Figure 5.4: Position and energy data of the fourth experiment, which covers a normal simulation including the virtual wall, that is, fault scenario I: Unexpected Obstacle. The graphs correspond to the simulation screenshots in Figure 4.18.

Figure 5.4a shows the three stages of the alternative trajectory described in Section 4.2.5. It also shows that the robot's end effector has a distance of about 0.1 m to the wall when the robot hits it. Figure 5.4b shows that the collision occurs at $t = 3.18$ s (sample index 319) and is detected at $t = 3.21$ s (sample index 322). Hence, the setpoint generator detected the collision after 3 time samples, corresponding to 30 ms. Figure 5.4c shows that a total of 132.8 J is provided by the energy estimator and a total of 131.9 J is consumed by the loop controller. Furthermore, it shows that, after the collision, the energy-tank level (equal to the difference between provided and consumed energy) rises above average until approximately $t = 4$ s. Figure 5.4c also shows that the provided energy budgets, up until the collision, are near equal to those of the third experiment shown in Figure 5.3c. Figure 5.4d shows that the potential energy gradually increases after the collision while the kinetic energy experiences a more rapid decline. It also shows that the tank and potential energies experience a turn in rate of change one time sample after the collision detection.

5.3 Discussion

This section discusses the design and implementation of the energy-aware sequence controller by evaluating them at the hand of the experiment results presented in Section 5.2. First, each of the three sequence-control components (setpoint generator, energy estimator, and passivity layer) is treated, followed by the three system-level properties that have also been used in Section 3.1 to analyse other implementations of sequence control with.

5.3.1 Sequence-Control Components

This section evaluates to what extend the setpoint generator, energy estimator, and passivity layer have achieved their functionalities described in Section 4.2.

Setpoint Generator

In Section 4.2.5, it is claimed that the setpoint generator outputs setpoints in straight line trajectories between targets, adapts the setpoint velocity when the energy-tank level reaches low values, and performs collision detection and correction. Figures 5.1a, 5.2a, 5.3a and 5.4a confirm the straight line trajectories. Figure 5.1 shows that the setpoint velocity indeed reduces as a function of energy tank depletion, thereby maintaining passivity (the amount of consumed energy never exceeds the amount of provided energy) and preventing the passivity layer from triggering—which, according to Brodskiy (2014), would otherwise negatively impact system performance. This already shows that the implemented sequence controller is energy-aware.

In Section 5.2.1 of experiment 1, it is noted that the robot (at $x = -0.4155$ m) does not fully reach the setpoint (at $x = -0.4159$ m) in Figures 5.1a and 5.1b. This is because the combination of virtual damping and a low (and decreasing) virtual-spring force in the loop controller inhibits robot movement.

In Section 5.2.2 of experiment 2, it is noted that Figure 5.2b shows that the setpoint motion speed at the start of the simulation is lower than the the specified 0.75 m/s. This is because the setpoint also has a y- and z-component when moving from the initial position to target A, whereas Figure 5.2b only considers the x-axis. The Euclidean setpoint velocity is in fact 0.75 m/s between targets throughout experiment 2. Also noted for Figure 5.2b, is that targets A and B are at $x = 0.548$ m and $x = -0.547$ m, respectively, instead of precisely at 0.55 m and -0.55 m specified in Section 4.2.4. This is because the setpoint generator assumes having reached a target as soon as the distance between the target and the setpoint is less than the step size in between subsequent setpoints.

In Section 4.2.4 it is explained that the supervisory controller checks if the robot has reached a target before switching to the next target. However, since higher control levels should deal with higher abstractions from the physical world, in future implementations of energy-aware sequence control, assessment of the robot state should be performed by the setpoint generator instead. The setpoint generator can then write feedback to the supervisory-control layer on the progress of task execution.

From Figure 5.4d of experiment 4, it can be seen that the kinetic and potential energies react quicker and more distinguishably to a collision than the energy-tank level (which experiences a more gradual decline). Be aware that the sudden increase of the energy-tank level at $t = 3.22$ s is not an effect of the collision, but of the change in setpoint direction (explained below for the energy estimator). This marks the importance of using loop-control feedback energies rather than the energy-tank level for the assessment of physical interactions. As a result, the setpoint generator takes 3 time samples—equivalent to 30 ms (since the sequence controller runs at 100 Hz)—to detect a collision and start performing its correction sequence. In future implementations for which the speed of physical interaction assessments is critical, as could

be the case for collision detection, such functionalities may need to be implemented under a faster update frequency, or (for the same effect) possibly at the loop-control layer.

Figure 5.4d also shows that, after the collision, the potential and kinetic energies experience a change in opposite direction. This shows the importance of assessing these energies separately rather than their sum. Be aware that the kinetic energy experiences a more rapid change than the potential energy only because the robot hits the wall perpendicularly. How each energy type responds to a physical interaction differs per situation. Therefore, the results in Figure 5.4d only tell that, depending on robot velocity, kinetic energy is a more discernible indicator of collisions that occur perpendicularly than potential energy is.

The architecture design is implementation-agnostic as it allows the use of any of the four physical variables listed in Tables A.1 and A.2, out of which the setpoint generator implementation has been made functional with H-matrices. Since the setpoint generator implementation uses H-matrices and only considers xyz-coordinates and disregards rotations, it is not implementation-agnostic. The setpoint generator design, however, is implementation-agnostic as it allows for any implementation of the setpoint generator that satisfies the design and implementation requirements of Section 3.5. In Section 4.1.3 it is stated that the setpoint generator is to satisfy Design Requirement 5, which specifies that it must send setpoint commands based on energy feedback. This has been achieved in the implementation, as shown in this chapter.

Energy Estimator

In Section 4.2.6 it is stated that the energy estimator provides energy budgets of 0 J when the setpoint velocity is zero and otherwise applies Equation 4.3 while keeping a minimum energy budget of 0.01 J. The minimum energy budget is to prevent negative energy budgets and the possibility of a deadlock in which the setpoint generator, energy estimator, passivity layer, and robot are all waiting for each other to move. Note should be taken that this deadlock is specific to the implementation in this project. Its presence may not exist in implementations where the setpoint generator does not adjust the setpoint velocity based on energy-tank feedback, or where the energy estimator does not output energy budgets on the basis of robot position and orientation feedback. Still, it is advisable to consider the risk of such deadlocks during the design and implementation phases in future work.

Additionally, the deadlock prevention measure of forcing a minimum energy budget is a sub-optimal solution that, depending on the type of implementation, can potentially lead to non-passive behaviour. Instead, an energy estimator could include (potentially complex) algorithms that predict a deadlock and prevent it from taking place.

Figures 5.3c and 5.4c support that, indeed, no negative energy budgets are outputted. These figures also show periods during which the provided energy budget has a constant value of 0.096 J, most notably at the start of the simulation and right before and after periods of 0 J. The average energy value in the energy-budget log file is $E_{\text{average}} = 0.192$ J, the energy budgets of 0.096 J result from Equation 4.3 for $E_{\text{linenumber}} = 0$ J. It makes sense that this happens right before and after periods of 0 J energy budgets, because these occur near the target positions where the setpoint velocity equals zero.

However, the period of multiple 0.096 J energy budgets at the start of the simulation should not be taking place. It is the result of insufficient starting energy combined with a low setpoint velocity caused by the initial empty energy tank. This then leads to slower robot movement and a different robot rotation compared to the budget-generation simulation of experiment 2. Due to the different robot movement, H-matrix row-index detection stays at row-index 1 (of 0 J) at the start of the simulation longer than it should be. A cause of the problem is the lower energy-budget amplitude resulting from Equation 4.3, which makes it inherently impossible to have experiments 3 and 4 achieve the same results as in experiment 2. A possible solution to the lack

of energy at the start of the simulation could be to provide packets of multiple energy budgets in advance instead of providing a single energy budget for each upcoming iteration separately.

Figure 5.2c of the budget-generation experiment (experiment 2) shows three sections of negative sampled energy budgets with peak values of -0.201, -0.207, and -0.190 J, respectively. These negative energy values indicate a transfer of energy from the robot into the energy tank. This is caused by opposite directions of the setpoint velocity and the robot's generalised forces (Equation A.2). The result is that $E_{\text{linenumber}}$ of Equation 4.3 obtains these negative values when the energy budgets are being read from the log file. Given that $E_{\text{average}} = 0.192$ J, this then yields $E_{\text{estimated}} < E_{\text{min}} = 0.01$ J for all three negative peaks. When this happens, the energy estimator should output $E_{\text{min}} = 0.01$ J instead of $E_{\text{estimated}}$. Figure 5.3 of experiment 3 indeed shows that between $t = 4.47$ s and $t = 4.64$ s, the minimum energy value of $E_{\text{min}} = 0.01$ J is outputted.

In Section 5.2.3 on experiment 3, it is mentioned that the energy estimator provided a total of 134 J over the 7.56 s simulation. This should have been equal to the total energy consumed during experiment 2 in Section 5.2.2 (i.e. 171.3 J), were it not that the robot in experiment 3 covered less distance in the same amount of time. The amount of energy consumed during experiment 2 for that same distance is 140.6 J and was achieved at $t = 6.16$ s. Hence, experiment 3 took 6.6 J less energy to cover the same distance while taking 1.4 s more time. This indicates that, for a normal simulation (experiment 3) under the same conditions as the budget-generation simulation (experiment 2), the total energy outputted by the energy estimator over a certain path—rather than over the same amount of time—is close to what it should be (within 5% in this case). This makes sense, because the energy estimator provides energy budgets based on the robot's pose regardless of time.

From Figure 5.4c of experiment 4, it can be seen that, following the collision, the energy-tank level increases above average. When the setpoint generator detects the collision and reverses the setpoint's direction, the virtual spring (temporarily) relaxes, allowing potential energy to flow back into the energy tank. During this relaxation time—in which no (or less) additional energy is needed—the energy estimator continues providing energy budgets based on the robot's position. This results in an offset in the energy-tank level. The only reason that the energy-tank level offset is later depleted from the energy tank, is because the alternative trajectory requires more energy per time step than the energy estimator is outputting. Hence, the implemented energy estimator does not provide adequate energy budgets following a collision and during an alternative trajectory.

The slower the setpoint generator is at detecting a collision, the higher the energy-tank level offset becomes. Depending on how high the offset is and how much additional energy is required for the alternative trajectory, the energy tank's offset may not be fully depleted. In such a case, each wall collision would further increase the energy tank's offset. Other scenarios that can lead to an increasing offset in the energy tank are when the control system interacts with an active environment or with another (physical) system. These interactions can lead to a transfer of energy into the sequence controller's energy tank, raising its level.

Since it is the energy estimator's responsibility to ensure that adequate energy is present in the energy tank for system task performance, future implementations could make use of an energy-tank feedback signal from the passivity layer to the energy estimator. The energy estimator may then adjust its energy-budget provision based on the amount of energy present in the passivity layer's energy tank. This could prevent any excessive energy build-up in the energy tank, potentially preventing unsafe high-energy actions.

In Section 4.1.3 it is stated that the designed energy estimator is to satisfy Design Requirement 6b, i.e. it should output accurate energy-budget estimations. This chapter has shown that the implemented energy estimator does not produce fully accurate energy-budget estimations, but

is sufficient in demonstrating energy awareness of the designed and implemented sequence-control layer.

Additionally, in future implementations that include an energy supervisor or higher control layers, energy estimation may be removed from the sequence-control architecture. Energy budgets do need to be provided to the passivity layer, but this may also be done through the energy supervisor. Energy estimation can then be conducted at higher control levels such as the supervisory-control layer or perhaps by the energy supervisor itself.

Passivity Layer

Because the setpoint generator applies setpoint-speed reduction based on the energy-tank level, it has successfully prevented the passivity layer from triggering during any of the experiments. Since the passivity layer's implementation is based on proven concepts, no additional data results are given to show its performance. The passivity layer has been tested for this project, though, and it is functional.

Figures 5.1c, 5.3c and 5.4c, show passive behaviour, because the total consumed energy does not exceed the total provided energy or, equivalently, the energy-tank level does not become negative. Passivity, in these cases, is maintained by the setpoint generator reducing the setpoint velocity as a function of the energy-tank level. This functionality effectively prevents the passivity layer from taking control over the setpoint command, keeping control at the setpoint generator. The setpoint generator here can be seen as a first measure to softly aid passivity while the passivity layer is a second measure that brute-forces passivity if the setpoint generator fails to do so. Hence, the implementation uses two separate functionalities that aid to maintain system passivity.

In Section 4.2.7 it is claimed that the implemented passivity layer can calculate energy exchanges with Equation 3.5 from any of the combinations listed in Table 3.1 in the form of variables from the mechanical domain listed in Table A.1 and from screw theory listed in Table A.2. However, the passivity layer has only been tested with H-matrix command signals and wrench-feedback signals. Other signal combinations are yet to be tested. In Section 4.1.3 it is stated that the passivity layer is to satisfy Design Requirement 6a of incorporating a (possibly reused) passivity layer. This requirement has been satisfied.

5.3.2 System-Level Properties

This section covers Implementation Requirement 2, that is, this project's implemented energy-aware sequence-control layer is compared to the six projects listed in Section 3.1 on the basis of system-level properties safety, dependability and performance.

Safety

Sequence-control implementations analyses in Section 3.1 mention that passivity leads to stability and thus safety, also when handling an unknown environment. In this project, there have indeed been no stability issues. Safety has also been improved, because Figure 5.1 shows that the robot will gradually stop moving as the energy tank deprives from energy. Furthermore, Figure 5.4c shows that a break of passivity and unsafe behaviour have not been at risk following the collision with the unknown obstacle because of timely detection and reaction.

For the PIRATE, Pneumatic Drive, and Autonomous ATV sequence-control implementations that are analysed in Section 3.1, it is stated that energy-aware control can ensure passivity during communication delays and thereby ensure stability and safety. The effects of communication delays have not been tested in this project. Though, the passivity layer does use the real control update interval ($\mathrm{d}t_{\text{real}} = t_{\text{current}} - t_{\text{previous}}$) instead of the pre-specified update time ($\mathrm{d}t_{\text{specified}} = {f_{\text{specified}}}^{-1}$) when calculating energy consumptions, so passivity should be kept with communication delays.

Dependability

For the Robot Hand project analysed in Section 3.1, it is mentioned that energy awareness could improve reliability for handling unknown objects. Experiment 4 shown in Figure 5.4 gives an example scenario in which energy awareness indeed provides a reliable response for handling an unknown object. The response is reliable in the sense that there have been zero instances of false negatives in the setpoint generator's obstacle detection. False positives can occur, though, as discussed below for the system-level property of performance.

For the Exoskeleton implementation of sequence control that is analysed in Section 3.1, it is mentioned that dependability can be improved as an energy-aware controller can assess both static and dynamic situations through potential- and kinetic energies. This has also been shown in experiment 4, where the combination of potential- and kinetic energy fed back from the loop controller was used by the setpoint generator to successfully detect the collision.

Performance

Brodskiy (2014) stated that inaccurate energy-budget estimations negatively impact system performance. Figure 5.3b shows that the robot in experiment 3 covers less distance than in Figure 5.2b of experiment 2. Hence, enabling energy awareness resulted in a negative effect on the system's speed. As explained above, this is indeed the result of inaccurate energy-budget provisions by the energy estimator, thus confirming the statement by Brodskiy (2014) that inaccurate energy-budget estimations negatively impact system performance.

For the PIRATE, TUlip, and Exoskeleton sequence-control implementations that are analysed in Section 3.1, it is mentioned that energy-aware sequence control can improve system versatility. This has been demonstrated with experiment 4 in Figure 5.4, showing that energy-awareness can be used to perform collision detection within 30 ms. In future work, the energy assessing block in Figure 4.5 can be implemented such that it assesses a wide range of physical interactions on the basis of energy data, for example the fault scenarios presented in Figure 3.6.

A significantly disruptive factor experienced during development of the sequence controller has been hick-ups in the simulation, resulting in a worsening of ROS' non real-timeness (significantly inconsistent sample times) and incorrect discrete amplitudes of variables propagating through the entire control system (i.e. apparent in variables of the robot, loop controller, and sequence controller). For the budget-generation simulation, the hick-ups result in the logging of incorrect energy-budgets. For the setpoint generator, the hick-ups can lead to false positives in the collision detection algorithm. For the passivity layer, the hick-ups result in incorrect energy sampling (and, hence, incorrect energy-tank levels). This all has a negative impact on system safety, dependability and performance.

Even though hick-ups may occur in animation, in a fully simulated system these should not have affected the values of variables. This indicates that there may be flaws in how the simulation engines of ROS and Gazebo resolve their timing issues. It has been found that using 'best performance' mode on the host computer (as stated in Appendix F) results in more reliable simulations. This way, the simulations in Section 5.2 have been obtained without hick-ups. Note that a limitation on computing performance has been the use of a virtual machine to run Ubuntu on (Appendix F). To further reduce the computational load, it could also help to run the simulation without an animation, log the data to a file, and replay the animation afterwards.

Since all experiments have been performed in simulation, no real-world data has been obtained. System-level properties may deviate in practise, because of real-world uncertainties like friction and vibrations. The collision detection feature and energy-budget estimation, for example, have not been tested for robustness against these uncertainties and may perform differently in practise. Future implementations will have to be tested on a physical robot to yield results that give better insight in real-world performance of energy-aware sequence control.

6 Conclusions and Recommendations

6.1 Conclusions

The project presented in this thesis has focussed on integrating energy awareness into the sequence-control layer of the control stack shown in Figure 2.2. In Section 1.3, the project's main goal and its four subgoals are presented. Each of these goals is separately concluded upon below, starting with the subgoals leading up to the main goal.

The project's first subgoal,

identifying how energy awareness can be enabled and harnessed at the sequence-control layer,

has been achieved. To enable energy awareness, two physics-conformal interfaces for the communication of data within an energy-aware sequence-control layer have been proposed. In Section 3.2.1, a control interface is presented that incorporates each of the four physical variables: generalised force (effort), generalised momentum, generalised velocity (flow), and generalised displacement. Alongside the control interface, an energy interface that allows for the communication of 7 different energy variables is presented in Section 3.2.2. Using these interfaces, energy awareness—the planning and performing of actions taking into account a system's energetic state (Section 1.1)—can be enabled.

In Section 3.1 it is found that energy awareness at the sequence-control layer can lead to improvement of system-level properties, but does require accurate sensor and actuator calibrations, collocation of the control signals, and accurate energy budgets for the passivity layer's energy tank. To the end of harnessing energy awareness, Section 3.4 presents 5 fault scenarios in which energy awareness could be used to improve fault handling by a robotic control system. Out of these scenarios, one that consists of an unexpected obstacle has been implemented in this project. Experiment 4 in Chapter 5 successfully proves that energy awareness can be harnessed to perform obstacle-collision detection.

The project's second subgoal,

defining design and implementation requirements for energy-aware sequence control,

is achieved with a list of MoSCoW-prioritised requirements presented in Section 3.5. The requirements are based on background concepts of robot control and energy-related physics described in Chapter 2 and the analyses of energy awareness in Chapter 3. By satisfying the design and implementation requirements, an energy-aware sequence-control layer or any of its components can be developed and connected to elements that satisfy the same requirements.

The project's third subgoal,

designing and implementing the architecture of an energy-aware sequence-control layer,

is achieved in Chapter 4, in which a an implementation-agnostic design and conceptual implementation are presented based on the design and implementation requirements listed in Section 3.5. The design in Section 4.1 presents the architecture for an energy-aware sequence-control layer consisting of a control interface, energy interface, setpoint generator, energy estimator, and passivity layer. Future implementations of an energy-aware sequence-control layer or its component can be based on this architecture design. As a proof of concept, the design has conceptually been implemented in simulation as presented in Section 4.2.

The two interfaces provide a physics-conformal and energy-based means of communicating control and energy signals between components that interact with and within the sequence-control layer. The setpoint generator enables energy-aware control for the sequence-control

layer by assessing energetic states of the control system and adjusting the output setpoint trajectory accordingly. The energy estimator provides model-based energy budgets to the passivity layer's energy tank. The energy budgets correspond to amount of energy that the loop-control layer is expected to extract from the passivity layer for controlling the robot. It is important for the energy budgets to be accurately estimated, because they can otherwise have a significant negative effect on system performance, as is demonstrated by experiments 3 and 4 in Chapter 5. The passivity layer acts as a passivity safeguard that provides a passive connection between the sequence-control layer and the loop-control layer.

The project's fourth subgoal,

demonstrating improved system-level properties through energy awareness,

is achieved in Chapter 5, in which four experiments and their results are presented and evaluated. System-level properties of safety, dependability and performance have been evaluated in comparison to six other projects that implemented non-energy-aware sequence control and are each analysed in Section 3.1. It has been demonstrated that the energy-aware setpoint generator can be used to aid in maintaining system passivity, thereby reducing the chance of passivity-layer triggers. This benefits system performance as the passivity layer does not take over setpoint control from the setpoint generator. Since the system has been implemented in simulation, it is unknown how similar the system-level properties are for an implementation on a real-world system. Real-world uncertainties like friction and vibrations, for example, may affect the accuracy of energy-budget estimations and the performance of the setpoint generator's energy assessment. The effects of a real-world implementation are to be evaluated in future work.

The main goal of this project, as stated in Section 1.3, has been to take the next step in energy-aware robotics; namely,

enabling energy awareness at the sequence-control layer.

As can be concluded from the achievement of each of the four subgoals, the main goal of this project has been achieved with a full design, implementation, and evaluation of a sequence-control layer that is energy aware. By achieving all of its goals, the project presented in this thesis has taken the next step towards fully energy-aware robotics. It provides an architectural basis of energy-aware sequence control for further development of energy awareness at higher control layers in physically interacting robotic control systems.

6.2 Recommendations

Since, in this project, the energy-aware sequence controller's design has been implemented to serve only as a proof of concept, most of the recommendations for future work are focussed on the implementation. The recommendations are MoSCoW (Must, Should, Could, Will not) prioritised and grouped under the control system in general (Section 6.2.1), the setpoint generator (Section 6.2.2), the energy estimator (Section 6.2.3), and the passivity layer (Section 6.2.4). Note that any future work on the energy-aware sequence-control layer should be based on the architecture design presented in Section 4.1. As stated in Section 4.1.3, the energy-aware sequence control design presented in this thesis does not restrict future additions of new signals, as long as compliance to the control- and energy interfaces is maintained.

6.2.1 Control System

- **Must**
  Future work on the energy-aware sequence-control layer *must* be implemented on real-world robots and evaluated accordingly.

- **Should**
  In line with the plans of RaM described in Section 2.4, and included in the design of Section 4.1, future projects *should* implement an energy supervisor for the communication of energetic system data and the distribution of energies between control layers.

6.2.2 Setpoint Generator

- **Should**
  Future work on the setpoint generator *should* include functionality expansions of the energy assessing block such that it can assess a broader range of physical interactions.
- **Should**
  The setpoint generator *should* receive $robot_{\text{state}}$, check if targets are reached, and feed back the sequence-control layer's control state to the supervisory-control layer. This is contrary to the the implemented sequence-control layer shown in Figure 4.12, in which the supervisory-control layer receives feedback-signal $robot_{\text{state}}$ to determine if targets are reached.
- **Could**
  In future work, the assessment of energy data *could* be implemented at a higher (possibly different from the rest of sequence control) control frequency. Future work would have to point out if this frequency should be equal to the loop-control layer's frequency. This is because, for certain physical interactions (such as collisions), the speed of assessment and reaction can be critical from the perspective of system-level properties like safety.
- **Could**
  Future design and implementation *could* feed back the attenuated setpoint from the passivity layer to the setpoint generator as shown in Figure C.3, so that the setpoint generator is aware of the actual output to the loop controller and can synchronise its setpoint/trajectory state accordingly.

6.2.3 Energy Estimator

- **Must**
  Future energy-estimator implementations *must* be model-based. It is important for system performance that the energy estimator produces accurate energy budgets. Consequently, the models need to be sufficiently accurate too.
- **Should**
  Future implementations *should* make use of an energy-tank feedback signal to the energy estimator as shown in Figure C.3 (branching off from signal *tank* in Figure 4.3). The energy estimator can then adjust its energy-budget provision based on the energy-tank level. This is needed to prevent uncontrolled energy build-up in the energy tank.
- **Could**
  Future implementations *could* make use of algorithms that—by checking if the setpoint generator, energy estimator and passivity layer are all waiting for each other—predict and/or detect deadlocks and counteract them by adjusting the energy-budget supply. This could replace the potentially non-passive approach of forcing a minimum energy budget to prevent passivity-induced deadlocks used in this thesis.

6.2.4 Passivity Layer

- **Should**
  The passivity layer implementation in this project has only been tested for H-matrix commands with wrench feedback signals. In future implementations, the passivity layer *should* be tested for other input combinations as well.

A Basic Principles of Energy Exchange

The following corresponds to Section 2.3.

A.1 Quantities of Physical Interaction

Robotic systems use sensors to measure physical interactions. Depending on the type of sensor, quantities such as those listed in Table A.1 can be measured.

Table A.1: Quantities per physical domain (and screw theory) classified under four physical variables: flow, effort, generalised displacement, and generalised momentum (Breedveld, 1982)

physical domain	**flow**	**effort**	**generalised displacement**	**generalised momentum**
	f	e	$q = \int f \, \mathrm{d}t$	$p = \int e \, \mathrm{d}t$
mechanical translation	velocity	force	displacement	momentum
	v [ms^{-1}]	F [N]	x [m]	p [Ns]
mechanical rotation	angular velocity	torque	angular displacement	angular momentum
	ω [rad.s^{-1}]	τ [Nm]	θ [rad]	b [Nms]
electromagnetic	current	voltage	charge	magnetic flux linkage
	i [A]	u [V]	q [As]	λ [Vs]
hydraulic	volume flow	pressure	volume	momentum in a flow tube
	φ [m^{3}s^{-1}]	p [Nm^{-2}]	V [m^3]	Γ [Nm^{-2}s]

The quantities listed in Table A.1 are classified as a generalised velocity or flow (f), generalised force or effort (e), generalised displacement (q), or generalised momentum (p). These four physical variables are called the variables of state and can be used to describe the energetic condition of any physical state-determined system (Paynter, 1961). Their interrelations are shown in Figure A.1, in which the variables are written in vector/matrix notation. Vectors are used in multi-DOF robotics to contain multiple quantities of the same type. Consider a 4-DOF robot with a displacement sensor at each joint. For this robot, the generalised-displacement vector equals $\boldsymbol{q} = [q_1, q_2, q_3, q_4]^\top$. In the remainder of this thesis, the variables of state are considered for multi-DOF robotics, thus in vector form ($\boldsymbol{f}$, $\boldsymbol{e}$, $\boldsymbol{q}$, and $\boldsymbol{p}$).

A common method when working on robot kinematics is to apply screw theory, in which translational and rotational motions are combined to describe the dynamics of rigid body systems. In screw theory, the equivalents of generalised velocity, force, displacement, and moment, are the twist vector ($\boldsymbol{T} \in \mathbb{R}^{6\times1}$), wrench vector ($\boldsymbol{W} \in \mathbb{R}^{6\times1}$), homogeneous-transformation matrix (or H-matrix in short, $\boldsymbol{H} \in \mathbb{R}^{4\times4}$), and momenta ($\boldsymbol{N} \in \mathbb{R}^{6\times1}$), respectively (Stramigioli and Bruyninckx, 2001). Since the twist vector is of dimension 6×1 and the H-matrix is of dimension 4×4, the relation $\boldsymbol{q} = \int \boldsymbol{f} \, \mathrm{d}t$ does not hold true here. Instead, the time-dependent relation is

$$\tilde{\boldsymbol{T}}_i^{j,j} = \dot{\boldsymbol{H}}_i^j \boldsymbol{H}_j^i, \tag{A.1}$$

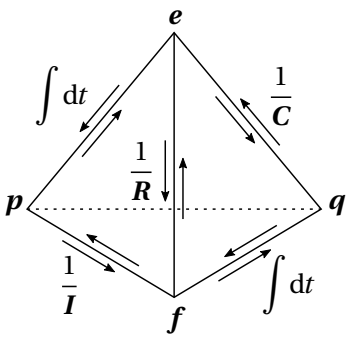


Figure A.1: The tetrahedron of state by Paynter (1961) describing the relations between the four physical variables; flow ($\boldsymbol{f}$), effort ($\boldsymbol{e}$), generalised displacement ($\boldsymbol{q}$) and generalised momentum ($\boldsymbol{p}$). The dynamic (time-dependent) interactions are embodied by the $\boldsymbol{q}$-$\boldsymbol{f}$ and $\boldsymbol{p}$-$\boldsymbol{e}$ relations. The remaining characteristic static relations are defined by inertance ($\boldsymbol{I}$), resistance ($\boldsymbol{R}$) and capacitance ($\boldsymbol{C}$).

and describes the twist matrix ($\tilde{\boldsymbol{T}}_i^{j,j} \in \mathbb{R}^{4\times4}$) of a body i with respect to a body j, expressed in the coordinate frame of body j. The variables that make up the corresponding twist vector ($\boldsymbol{T}_i^{j,j} \in \mathbb{R}^{6\times1}$) can directly be extracted from this twist matrix. Additional information on screw theory can be found in the work by Stramigioli and Bruyninckx (2001). Table A.2 is added as a screw theory extension of Table A.1.

Table A.2: Quantities of screw theory (Stramigioli and Bruyninckx, 2001) classified under the variables of state: flow, effort, generalised displacement, and generalised momentum.

domain	**flow**	**effort**	**generalised displacement**	**generalised momentum**
	$\boldsymbol{f}$	$\boldsymbol{e}$	$\boldsymbol{q}$	$\boldsymbol{p} = \int \boldsymbol{e}\,\mathrm{d}t$
screw theory	twist	wrench	H-matrix	momenta
	$\boldsymbol{T} \in \mathbb{R}^{6\times1}$	$\boldsymbol{W} \in \mathbb{R}^{6\times1}$	$\boldsymbol{H} \in \mathbb{R}^{4\times4}$	$\boldsymbol{N} \in \mathbb{R}^{6\times1}$

A.2 Energy-based Information Exchange

An effort does not exist without a flow and vice versa; effort and flow are power conjugated (Breedveld, 1985). The flow rate of energy ($\mathrm{d}E/\mathrm{d}t$), or instantaneous exchange power (P), can be calculated from the product of power-conjugated effort and flow:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}E &= P \\ &= \boldsymbol{e}^\top \cdot \boldsymbol{f}\,. \end{aligned} \tag{A.2}$$

The amount of energy (E) exchanged during a physical interaction is equal to the integration of energy flow, or power, over time:

$$\begin{aligned} E &= \int \left(\frac{\mathrm{d}}{\mathrm{d}t}E\right) \mathrm{d}t \\ &= \int (P)\ \mathrm{d}t \\ &= \int \left(\boldsymbol{e}^\top \cdot \boldsymbol{f}\right)\ \mathrm{d}t\,. \end{aligned} \tag{A.3}$$

Based on Equation A.3 (and Figure A.1), if a pair of power-conjugated effort (or generalised momentum) and flow (or generalised displacement) is known, the amount of energy exchanged over a certain time interval can be calculated. In practise, using a power-conjugated effort-flow pair implies the use of collocated control (Duindam et al., 2009), i.e. the actuation and measurement quantities need to be collocated.

In order to integrate energy awareness into the sequence-control layer, information exchange in and around the control system should be energy-related. This can be achieved by applying a port-based design. "In systems theory, the interconnection of two systems can be modeled by a bi-directional information (signals) exchange, which is termed a port. The energy exchange between these two systems will be a function of these signals in the port" (Brodskiy, 2014). Ports that convey a collocated pair of power-conjugated effort and flow are called power ports. Power ports can be connected using power bonds—from bond-graph theory by Paynter (1961). Both power ports and power bonds are power-continuous; they do not consume energy. Figure A.2 shows a generic example representation in which some entity (Entity 1) transfers power via its power port (A) through a power bond to the power port (B) of another entity (Entity 2). The entities can be systems, subsystems, digital components, physical components, etcetera, and can contain multiple power ports.

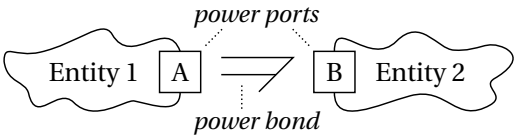


Figure A.2: Two entities with their power ports connected through a power bond.

Given that the design of the sequence-control level should follow an energy-based approach and given that power ports combined with power bonds facilitate lossless energy-flow interconnections, power ports and power bonds can be used to interconnect (sub)systems in and around the sequence-control layer. This does imply that each power connection in the system would require a known effort and flow.

Consider the plant (see Figure 2.2), its actuators receive a command signal coming from the loop-control layer and its sensors return a feedback signal to the loop-control layer. In order to be compliant to the power-bond connections, the command and feedback signals should comprise a power-conjugated effort-flow pair. Entities that receive a flow input and return a power-conjugated effort output resemble an impedance ($\boldsymbol{Z}$);

$$\boldsymbol{Z} = \frac{\boldsymbol{e}}{\boldsymbol{f}} \qquad \Rightarrow \qquad \boldsymbol{e} = \boldsymbol{f}\boldsymbol{Z}\,, \tag{A.4}$$

whereas entities that receive an effort input and return a power-conjugated flow output resemble an admittance ($\boldsymbol{Y}$);

$$\boldsymbol{Y} = \boldsymbol{Z}^{-1} = \frac{\boldsymbol{f}}{\boldsymbol{e}} \qquad \Rightarrow \qquad \boldsymbol{f} = \boldsymbol{e}\boldsymbol{Y}\,, \tag{A.5}$$

as shown in block diagram form in Figure A.3.

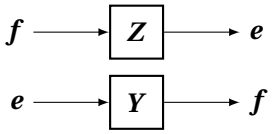


Figure A.3: Block diagram on the effort and flow relations of impedance ($\boldsymbol{Z}$) and admittance ($\boldsymbol{Y}$).

An impedance output can serve as an admittance input and vice versa. Impedances and admittances can be used to describe a control layer's conversion of the conjugated power variables.

B Equations

B.1 Twist Matrices and Vectors

The following corresponds to Section 3.3.1.

In general form, the twist matrix ($\tilde{\boldsymbol{T}} \in \mathbb{R}^{4\times4}$) and twist vector ($\boldsymbol{T} \in \mathbb{R}^{6\times1}$) are defined as follows (Stramigioli and Bruyninckx, 2001):

$$\tilde{\boldsymbol{T}} = \begin{bmatrix} \tilde{\boldsymbol{\omega}} & \boldsymbol{v} \\ \boldsymbol{0} & 0 \end{bmatrix}, \qquad \boldsymbol{T} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix}, \tag{B.1}$$

and consist of an angular velocity in tilde form ($\tilde{\boldsymbol{\omega}} \in \mathbb{R}^{3\times3}$), an angular-velocity vector ($\boldsymbol{\omega} \in \mathbb{R}^{3\times1}$) and a translational-velocity vector ($\boldsymbol{v} \in \mathbb{R}^{3\times1}$). The angular velocity in tilde and vector form ($\tilde{\boldsymbol{\omega}}$ and $\boldsymbol{\omega}$, respectively) consist of the same elements:

$$\tilde{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}, \qquad \boldsymbol{\omega} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}. \tag{B.2}$$

Equations B.1 and B.2 indicate that a twist vector ($\boldsymbol{T} \in \mathbb{R}^{6\times1}$) can directly be extracted from a twist matrix ($\tilde{\boldsymbol{T}} \in \mathbb{R}^{4\times4}$) and vice versa.

B.2 Setpoint Attenuation by the Passivity Layer

The following corresponds to Section 4.1.6.

Equation 4.2 can be expanded as follows:

$$setpoint_{\text{pas},k} = \begin{bmatrix} \boldsymbol{e}_{\text{pas},k} \\ \boldsymbol{p}_{\text{pas},k} \\ \boldsymbol{f}_{\text{pas},k} \\ \boldsymbol{q}_{\text{pas},k} \end{bmatrix} = \begin{cases} setpoint_k = \left[\boldsymbol{e}_k, \boldsymbol{p}_k, \boldsymbol{f}_k, \boldsymbol{q}_k\right]^\top & E^*_{\text{tank},k+1} \geq 0 \\ setpoint_{\text{att},k} = \left[\boldsymbol{e}_{\text{att},k}, \boldsymbol{p}_{\text{att},k}, \boldsymbol{f}_{\text{att},k}, \boldsymbol{q}_{\text{att},k}\right]^\top & E^*_{\text{tank},k+1} < 0 \end{cases}. \tag{B.3}$$

It is important to note that, to output *one* setpoint command to the loop controller, in Equation B.3 *only one* of the command variables ($\boldsymbol{e}_{\text{pas},k}$, $\boldsymbol{p}_{\text{pas},k}$, $\boldsymbol{f}_{\text{pas},k}$, or $\boldsymbol{q}_{\text{pas},k}$) should be used (i.e. the others should be set to zero and/or not be updated). The attenuated setpoint command ($setpoint_{\text{att}}$) used in Equation 4.2 (or Equation B.3) is defined such that Equation 4.1 results in $E^*_{\text{tank},k+1} = 0$ if $E_{\text{tank},k} > 0$ or in $E^*_{\text{tank},k+1} = E_{\text{tank},k}$ if $E_{\text{tank},k} \leq 0$:

$$setpoint_{\text{att},k} = \begin{bmatrix} \boldsymbol{e}_{\text{att},k} \\ \boldsymbol{p}_{\text{att},k} \\ \boldsymbol{f}_{\text{att},k} \\ \boldsymbol{q}_{\text{att},k} \end{bmatrix} = \begin{cases} \left[0, \boldsymbol{p}_{\text{pas},k-1}, 0, \boldsymbol{q}_{\text{pas},k-1}\right]^\top & E_{\text{tank},k} \leq 0 \\ \begin{bmatrix} \boldsymbol{e}_k \cdot \dfrac{E_{\text{tank},k}}{\Delta E^*_{\text{req},k+1}} \\ \boldsymbol{p}_{\text{pas},k-1} + \left(\boldsymbol{p}_k - \boldsymbol{p}_{\text{pas},k-1}\right) \cdot \dfrac{E_{\text{tank},k}}{\Delta E^*_{\text{req},k+1}} \\ \boldsymbol{f}_k \cdot \dfrac{E_{\text{tank},k}}{\Delta E^*_{\text{req},k+1}} \\ \boldsymbol{q}_{\text{pas},k-1} + \left(\boldsymbol{q}_k - \boldsymbol{q}_{\text{pas},k-1}\right) \cdot \dfrac{E_{\text{tank},k}}{\Delta E^*_{\text{req},k+1}} \end{bmatrix} & E_{\text{tank},k} > 0 \end{cases}. \tag{B.4}$$

Below follows the derivation of Equation B.4. Be aware that the derivations make use of Equation 3.5 in which $setpoint_{\text{att}}$ represents a command signal.

1. Derivation of $\boldsymbol{e}_{\text{att},k}$ (also holds for $\boldsymbol{f}_{\text{att},k}$):

   (a) If $E_{\text{tank},k} \leq 0$:
   The energy tank cannot be drained any further, i.e. Equation 4.1 shows that the energy requirement for upcoming sample interval $[t_k, t_{k+1}]$ should equal $\Delta E_{\text{req},k+1} = 0$. Inserting in Equation 3.5 yields:

   $$\Delta E_{\text{req},k+1} = \boldsymbol{e}_k{}^{\top} \cdot \frac{1}{2}\left(\boldsymbol{f}_{k+1} + \boldsymbol{f}_k\right) \cdot \Delta t_{k+1} = 0\,, \tag{B.5}$$

   and

   $$\Delta E_{\text{req},k+1} = \boldsymbol{e}_k{}^{\top} \cdot \left(\boldsymbol{q}_{k+1} - \boldsymbol{q}_k\right) = 0\,. \tag{B.6}$$

   Solving for the estimated required energy, $\Delta E^*_{\text{req},k+1}$, and the attenuated setpoint signal, $\boldsymbol{e}_{\text{att},k}$:

   $$\begin{aligned} \boldsymbol{e}_{\text{att},k}{}^{\top} \cdot \frac{1}{2}\left(\boldsymbol{f}^*_{k+1} + \boldsymbol{f}_k\right) \cdot \Delta t^*_{k+1} &= 0 \\ \boldsymbol{e}_{\text{att},k} &= 0 \end{aligned}\,, \tag{B.7}$$

   and

   $$\begin{aligned} \boldsymbol{e}_{\text{att},k}{}^{\top} \cdot \left(\boldsymbol{q}^*_{k+1} - \boldsymbol{q}_k\right) &= 0 \\ \boldsymbol{e}_{\text{att},k} &= 0\,. \end{aligned} \tag{B.8}$$

   (b) If $E_{\text{tank},k} > 0$:
   No more than the available energy can be drained from the energy tank, i.e. Equation 4.1 shows that the energy requirement for upcoming sample interval $[t_k, t_{k+1}]$ should equal $\Delta E_{\text{req},k+1} = E_{\text{tank},k}$. Again, inserting in Equation 3.5 and solving for the estimated required energy, $\Delta E^*_{\text{req},k+1}$, and the attenuated setpoint signal, $\boldsymbol{e}_{\text{att},k}$:

   $$\begin{aligned} \boldsymbol{e}_{\text{att},k}{}^{\top} \cdot \frac{1}{2}\left(\boldsymbol{f}^*_{k+1} + \boldsymbol{f}_k\right) \cdot \Delta t^*_{k+1} &= E_{\text{tank},k} \\ \boldsymbol{e}_{\text{att},k}{}^{\top} &= \frac{E_{\text{tank},k}}{\frac{1}{2}\left(\boldsymbol{f}^*_{k+1} + \boldsymbol{f}_k\right) \cdot \Delta t^*_{k+1}} \\ \boldsymbol{e}_{\text{att},k}{}^{\top} &= \frac{\boldsymbol{e}_k{}^{\top}}{\boldsymbol{e}_k{}^{\top}} \cdot \frac{E_{\text{tank},k}}{\frac{1}{2}\left(\boldsymbol{f}^*_{k+1} + \boldsymbol{f}_k\right) \cdot \Delta t^*_{k+1}} \\ \boldsymbol{e}_{\text{att},k}{}^{\top} &= \boldsymbol{e}_k{}^{\top} \cdot \frac{E_{\text{tank},k}}{\Delta E^*_{\text{req},k+1}} \\ \boldsymbol{e}_{\text{att},k} &= \boldsymbol{e}_k \cdot \frac{E_{\text{tank},k}}{\Delta E^*_{\text{req},k+1}}\,, \end{aligned} \tag{B.9}$$

and, equivalently,

$$\begin{aligned} \boldsymbol{e}_{\text{att},k}{}^{\top} \cdot \left(\boldsymbol{q}^*_{k+1} - \boldsymbol{q}_k\right) &= E_{\text{tank},k} \\ \boldsymbol{e}_{\text{att},k}{}^{\top} &= \frac{E_{\text{tank},k}}{\boldsymbol{q}^*_{k+1} - \boldsymbol{q}_k} \\ \boldsymbol{e}_{\text{att},k}{}^{\top} &= \frac{\boldsymbol{e}_k{}^{\top}}{\boldsymbol{e}_k{}^{\top}} \cdot \frac{E_{\text{tank},k}}{\boldsymbol{q}^*_{k+1} - \boldsymbol{q}_k} \\ \boldsymbol{e}_{\text{att},k}{}^{\top} &= \boldsymbol{e}_k{}^{\top} \cdot \frac{E_{\text{tank},k}}{\Delta E^*_{\text{req},k+1}} \\ \boldsymbol{e}_{\text{att},k} &= \boldsymbol{e}_k \cdot \frac{E_{\text{tank},k}}{\Delta E^*_{\text{req},k+1}} . \end{aligned} \tag{B.10}$$

2. Derivation of $\boldsymbol{p}_{\text{att},k}$ (also holds for $\boldsymbol{q}_{\text{att},k}$):

   (a) If $E_{\text{tank},k} \leq 0$:
   The energy tank cannot be drained any further, i.e. Equation 4.1 shows that the energy requirement for upcoming sample interval $[t_k, t_{k+1}]$ should equal $\Delta E_{\text{req},k+1} = 0$. Inserting in Equation 3.5 yields:

$$\Delta E_{\text{req},k+1} = \left(\boldsymbol{p}_{k+1} - \boldsymbol{p}_k\right)^{\top} \cdot \boldsymbol{f}_{k+1} = 0 , \tag{B.11}$$

   and

$$\Delta E_{\text{req},k+1} = \left(\boldsymbol{p}_{k+1} - \boldsymbol{p}_k\right)^{\top} \cdot \left(\boldsymbol{q}_{k+1} - \boldsymbol{q}_k\right) \cdot \frac{1}{\Delta t_{k+1}} = 0 . \tag{B.12}$$

   This is problematic, because we are working with the current setpoint value, $\boldsymbol{p}_k$, but Equations B.11 and B.12 also depend on $\boldsymbol{p}_{k+1}$, whose value is yet unknown and also depends on $\boldsymbol{p}_k$. A predicted value $\boldsymbol{p}^*_{k+1}$ is needed. The predicted setpoint change during sample interval $[t_k, t_{k+1}]$ can be defined as

$$\Delta \boldsymbol{p}^*_{k+1} = \boldsymbol{p}^*_{k+1} - \boldsymbol{p}_k . \tag{B.13}$$

   A first-order approximation can be used (if needed, a higher order approximation could be used) by estimating that the next setpoint change, $\Delta \boldsymbol{p}^*_{k+1}$, will be equal to the previous one, i.e. $\Delta \boldsymbol{p}^*_{k+1} = \Delta \boldsymbol{p}_k = \boldsymbol{p}_k - \boldsymbol{p}_{\text{pas},k-1}$. The estimated required energy then becomes

$$\Delta E^*_{\text{req},k+1} = \left(\boldsymbol{p}_k - \boldsymbol{p}_{\text{pas},k-1}\right)^{\top} \cdot \boldsymbol{f}^*_{k+1} = 0 , \tag{B.14}$$

   and

$$\Delta E^*_{\text{req},k+1} = \left(\boldsymbol{p}_k - \boldsymbol{p}_{\text{pas},k-1}\right)^{\top} \cdot \left(\boldsymbol{q}^*_{k+1} - \boldsymbol{q}_k\right) \cdot \frac{1}{\Delta t_{k+1}} = 0 . \tag{B.15}$$

   Solving for the attenuated setpoint signal, $\boldsymbol{p}_{\text{att},k}$:

$$\begin{aligned} \left(\boldsymbol{p}_{\text{att},k} - \boldsymbol{p}_{\text{pas},k-1}\right)^{\top} \cdot \boldsymbol{f}^*_{k+1} &= 0 \\ \boldsymbol{p}_{\text{att},k} &= \boldsymbol{p}_{\text{pas},k-1} , \end{aligned} \tag{B.16}$$

   and, equivalently,

$$\begin{aligned} \left(\boldsymbol{p}_{\text{att},k} - \boldsymbol{p}_{\text{pas},k-1}\right)^{\top} \cdot \left(\boldsymbol{q}_{k+1} - \boldsymbol{q}_k\right) \cdot \frac{1}{\Delta t_{k+1}} &= 0 \\ \boldsymbol{p}_{\text{att},k} &= \boldsymbol{p}_{\text{pas},k-1} . \end{aligned} \tag{B.17}$$

(b) If $E_{\text{tank},k} > 0$:
No more than the available energy can be drained from the energy tank, i.e. Equation 4.1 shows that the upcoming energy requirements should equal $\Delta E_{\text{req},k+1} = E_{\text{tank},k}$. Again, inserting in Equation 3.5, predicting $\Delta \boldsymbol{p}^*_{k+1} = \Delta \boldsymbol{p}_k$, and solving for the attenuated setpoint signal, $\boldsymbol{p}_{\text{att},k}$:

$$\begin{aligned}
\left(\boldsymbol{p}_{\text{att},k} - \boldsymbol{p}_{\text{pas},k-1}\right)^\top \cdot \boldsymbol{f}^*_{k+1} &= E_{\text{tank},k} \\
{\boldsymbol{p}_{\text{att},k}}^\top &= {\boldsymbol{p}_{\text{pas},k-1}}^\top + \frac{E_{\text{tank},k}}{\boldsymbol{f}^*_{k+1}} \\
{\boldsymbol{p}_{\text{att},k}}^\top &= {\boldsymbol{p}_{\text{pas},k-1}}^\top + \frac{\left(\boldsymbol{p}_k - \boldsymbol{p}_{\text{pas},k-1}\right)^\top}{\left(\boldsymbol{p}_k - \boldsymbol{p}_{\text{pas},k-1}\right)^\top} \cdot \frac{E_{\text{tank},k}}{\boldsymbol{f}^*_{k+1}} \\
{\boldsymbol{p}_{\text{att},k}}^\top &= {\boldsymbol{p}_{\text{pas},k-1}}^\top + \left(\boldsymbol{p}_k - \boldsymbol{p}_{\text{pas},k-1}\right)^\top \cdot \frac{E_{\text{tank},k}}{\Delta E^*_{\text{req},k+1}} \\
\boldsymbol{p}_{\text{att},k} &= \boldsymbol{p}_{\text{pas},k-1} + \left(\boldsymbol{p}_k - \boldsymbol{p}_{\text{pas},k-1}\right) \cdot \frac{E_{\text{tank},k}}{\Delta E^*_{\text{req},k+1}},
\end{aligned} \tag{B.18}$$

and, equivalently,

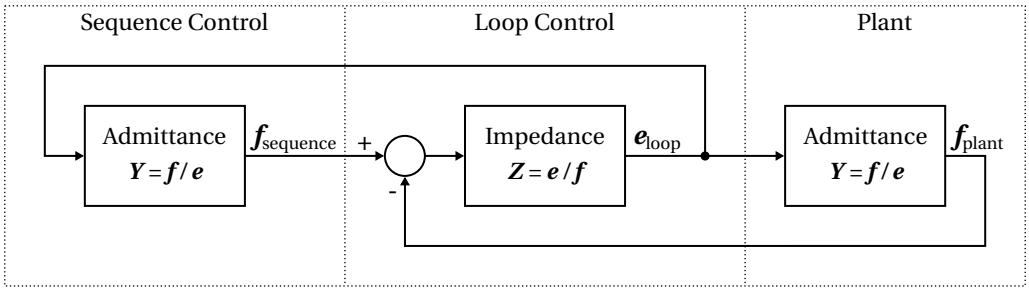
$$\begin{aligned}
\left(\boldsymbol{p}_{\text{att},k} - \boldsymbol{p}_{\text{pas},k-1}\right)^\top \cdot \left(\boldsymbol{q}^*_{k+1} - \boldsymbol{q}_k\right) \cdot \frac{1}{\Delta t_{k+1}} &= E_{\text{tank},k} \\
{\boldsymbol{p}_{\text{att},k}}^\top &= {\boldsymbol{p}_{\text{pas},k-1}}^\top + \frac{E_{\text{tank},k}}{\left(\boldsymbol{q}^*_{k+1} - \boldsymbol{q}_k\right) \cdot \frac{1}{\Delta t_{k+1}}} \\
{\boldsymbol{p}_{\text{att},k}}^\top &= {\boldsymbol{p}_{\text{pas},k-1}}^\top \\
&\quad + \frac{\left(\boldsymbol{p}_k - \boldsymbol{p}_{\text{pas},k-1}\right)^\top}{\left(\boldsymbol{p}_k - \boldsymbol{p}_{\text{pas},k-1}\right)^\top} \cdot \frac{E_{\text{tank},k}}{\left(\boldsymbol{q}^*_{k+1} - \boldsymbol{q}_k\right) \cdot \frac{1}{\Delta t_{k+1}}} \\
{\boldsymbol{p}_{\text{att},k}}^\top &= {\boldsymbol{p}_{\text{pas},k-1}}^\top + \left(\boldsymbol{p}_k - \boldsymbol{p}_{\text{pas},k-1}\right)^\top \cdot \frac{E_{\text{tank},k}}{\Delta E^*_{\text{req},k+1}} \\
\boldsymbol{p}_{\text{att},k} &= \boldsymbol{p}_{\text{pas},k-1} + \left(\boldsymbol{p}_k - \boldsymbol{p}_{\text{pas},k-1}\right) \cdot \frac{E_{\text{tank},k}}{\Delta E^*_{\text{req},k+1}}.
\end{aligned} \tag{B.19}$$

C Models

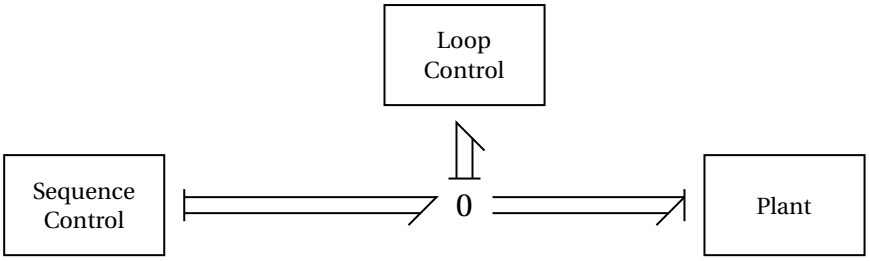
C.1 Impedance and Admittance Control

The following corresponds to Section 3.2.1.

Like in the examples of Section 3.1, a robot's actuators are often effort (torque or force) controlled and its sensors often measure flows (velocities or positions—from which velocities can be calculated). Hence, it is often the case that the robot's flow changes based on input efforts, i.e. that the robot (plant) resembles an admittance (see Section A.2). Consequently, the loop controller connected to the plant should receive flow input signals and return effort output signals, i.e. resemble an impedance. Then, the sequence controller commanding the loop controller should receive effort input signals and return flow output signals, i.e. resemble an admittance. This scenario, in which a loop controller calculates an effort command ($\boldsymbol{e}_{\text{loop}}$) based on the difference between a commanded sequence-control flow ($\boldsymbol{f}_{\text{sequence}}$) and a feedback plant flow ($\boldsymbol{f}_{\text{plant}}$), is shown in Figure C.1 in block diagram and equivalent bond-graph representations.



(a) Block diagram of control layers resembling an impedance or admittance.



(b) Bond-graph equivalent of Figure C.1a.

Figure C.1: Equivalent block-diagram and bond-graph representations of a system consisting of power-connected sequence control admittance, loop-control impedance, and plant admittance.

In accordance with Figure C.1, the loop-control layer developed by RaM for RobMoSys (described in Section 2.4) does resemble an impedance connected to an admittance plant. To connect a sequence controller to this impedance loop-control layer, a straightforward method would be to develop an energy-aware sequence-control layer that resembles an admittance and also uses a power port-based interface. However, following the good practises in design (Section 2.1), for a sequence-control layer to be modular, compliant, and extensible, it should not be limited to resembling admittance functionality only. Rather, an energy-aware sequence-control layer would have to be developed such that it, depending on the system it is implemented in, can function either as impedance or admittance.

C.2 Design Layout

The following corresponds to Section 4.1.

Figure 4.2 shows the sequence-control layer design's top view and Figure 4.3 shows the design's detailed architecture. A step in between these two design figures, is the design shown in Figure C.2, which presents the global component and signal layout.

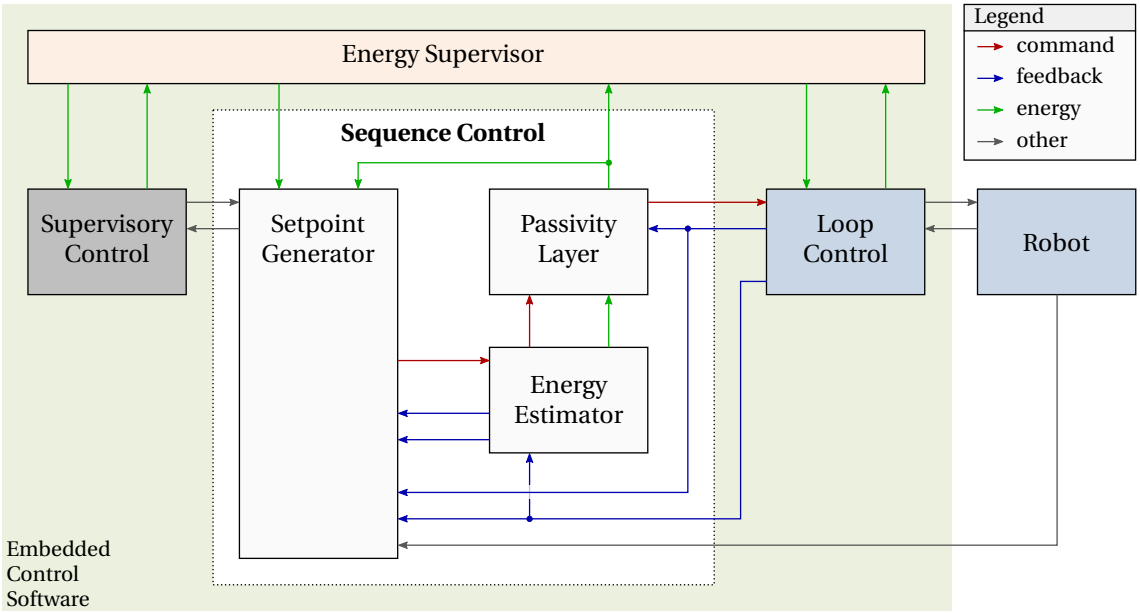


Figure C.2: Layout of the Sequence-Control layer design.

C.3 Recommendations – Additional Signals

The following corresponds to Section 6.2.

Two new signals are recommended in Section 6.2. The first recommended new signal is to feed back the passivity layer's output command, $setpoint_{\text{pas}}$, to the setpoint generator (recommended in Section 6.2.2). The second recommended new signal is to feed back the passivity layer's energy output—the energy-tank level contained by signal *tank*—to the energy estimator (recommended in Section 6.2.3). Both are illustrated in Figure C.3.

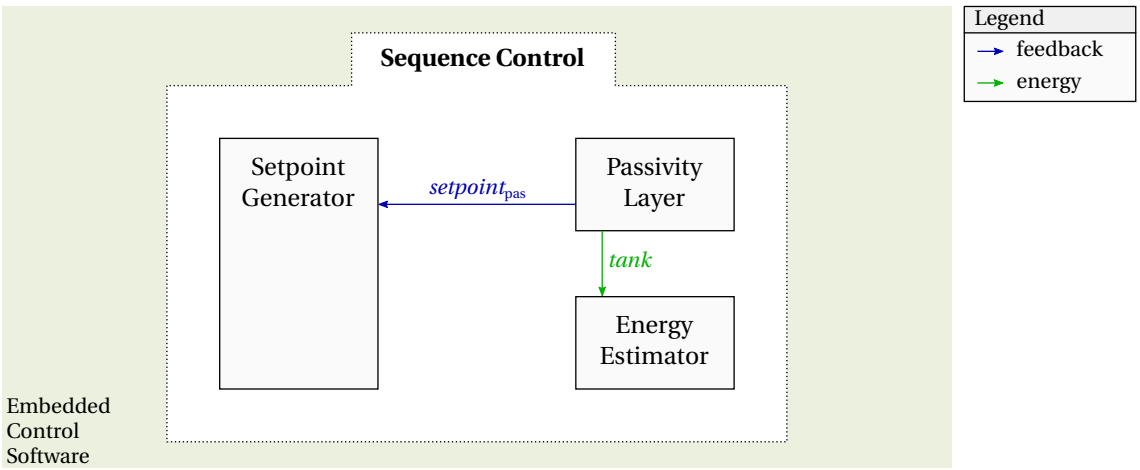


Figure C.3: Recommended additional signals described in Section 6.2.

D Implementation Details

D.1 Loop Control

The following corresponds to Section 4.2.1.

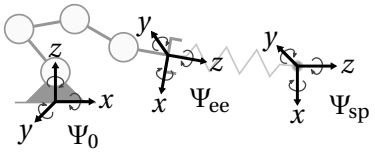


Figure D.1: Extension of Figure 3.4: illustrative placement of the inertial origin frame Ψ_0, robot end-effector frame Ψ_{ee}, and the setpoint frame Ψ_{sp}. A virtual spring modelled by the loop controller connects the robot end effector to the setpoint commanded by the sequence controller.

D.2 Interfaces

The following corresponds to Section 4.2.2.

The implemented interfaces described in Section 4.2.2 and shown in the UML class diagram of Figure 4.9 provide port variables that are private (-) and port functions that are either protected (#) or public (+). Since the port variables are private to the interfaces, they are not directly accessible to components that incorporate an interface, i.e. the setpoint generator has no direct access to the 'inputCommand' variable, for example. Instead, the port variables can only be indirectly accessed through 'getters' and 'setters' (get- and set-functions, respectively). The setters can be used to modify the value of a port variable. The getters can be used to retrieve the value of a port variable. The purpose hereof is to ensure correct handling of the communicated variables and prevent corruption due to (unintentional) misuse.

Public port functions are globally accessible for communication with external components. For example, the setpoint generator can access the public setter 'setInputCommand()' of the energy estimator. Protected port functions are only accessible to the component itself for internal usage. For example, the protected getter 'getInputCommand()' belonging to the energy estimator is accessible for the energy estimator itself, but not for the setpoint generator (which can only access its own 'getInputCommand()' function).

D.2.1 Control Interface

Based on Figure 4.10, consider the following example: the setpoint generator needs to transmit a setpoint command (`sp`) to the energy estimator. This is possible, because both components incorporate the control interface. There are two possible methods to transmit `sp` from the setpoint generator to the energy estimator:

1. Consider that the setpoint generator has a pointer to the energy estimator: `ptrEnergyEst`. The setpoint generator can write `sp` to the energy estimator by executing the following: `ptrEnergyEst->setInputCommand(sp)`. Hereafter, port variable `inputCommand` at the energy estimator contains the value of `sp`. Now, for the energy estimator's algorithm to access the setpoint and store it in a local variable, e.g. `spInput`, it can call the following: `spInput = this->getInputCommand()`. The `this`-pointer can also be omitted; it is merely included to clarify that the `getInputCommand()` function is called locally.
2. Consider that the energy estimator has a pointer to the setpoint generator: `ptrSetpointGen`. The setpoint generator can make `sp` available at its output by executing the following: `this->setOutputCommand(sp)`. Hereafter, the energy estim-

ator can read `sp` from the setpoint generator and store it in a local variable, e.g. `spInput`, by executing the following: `spInput = ptrSetpointGen->getOutputCommand()`.

Both of these two methods are used in the implementation, depending on which component has a pointer to another component. The setpoint generator, energy estimator, and passivity layer have been implemented hierarchically. The setpoint generator has a pointer to both the energy estimator and the passivity layer, the energy estimator has a pointer to the passivity layer, and the passivity layer has no pointers.

The control-signal type defined in Listing 4.1 is based on the four physical-variable type definitions in Listing D.1 below, which cover the physical quantities listed in Tables A.1 and A.2.

Listing D.1: Control-interface type definitions written in C++ of the four variables of state consisting of physical quantities listed in Tables A.1 and A.2. Note that `force`, `momentum`, `velocity` and `displacement` do not have a pre-determined size, because their sizes depends on the (unknown) amount of robot joints.

```
// Generalised force (effort)
struct eType {
    Matrix6x1 wrench{ Matrix6x1::Zero() };
    Eigen::VectorXd force{ Matrix2x1::Zero() };
};

// Generalised momentum
struct pType {
    Eigen::VectorXd momentum{ Matrix2x1::Zero() };
};

// Generalised velocity (flow)
struct fType {
    Matrix6x1 twist{ Matrix6x1::Zero() };
    Eigen::VectorXd velocity{ Matrix2x1::Zero() };
};

// Generalised displacement
struct qType {
    Matrix4x4 Hmatrix{ Matrix4x4::Identity() };
    Eigen::VectorXd displacement{ Matrix2x1::Zero() };
};
```

In Listing D.1, the `Matrix2x1`, `Matrix6x1` and `Matrix4x4` represent a 2×1 vector, 6×1 vector and 4×4 square matrix, respectively. These are defined using the Eigen[1] package as `typedef Eigen::Matrix<double,2,1> Matrix2x1`, `typedef Eigen::Matrix<double,6,1> Matrix6x1` and `typedef Eigen::Matrix<double,4,4> Matrix4x4`, respectively. The `Matrix2x1` is used to provide at least some initialisation to the vectors whose size depend on the amount of robot joints.

D.3 Supervisory Controller

The following corresponds to Section 4.2.4.

The implemented supervisory controller switches its output task between two targets of different x-position, A and B, in the form of an H-matrix. The robot arm has a length of 1.1976 m in fully extended upright position. The robot's bottom section has a height of approximately

[1] Eigen website: *http://eigen.tuxfamily.org/*

0.4 m, leaving about $1.1976 - 0.4 \approx 0.8$ m of the robot's body that can be extended into the xy-plane. Targets A and B are set at xyz-coordinates of $[0.55, 0.55, 0.4]$ m and $[-0.55, 0.55, 0.4]$ m, respectively. Hence, both targets are at a Euclidean xy-distance of 0.7778 m from the origin, falling within the robot's reach.

D.4 Energy Estimator

The following corresponds to Section 4.2.6.

To find the best-matching H-matrix, the energy estimator applies a weighting function. It checks which of the 16 elements in the 4x4 setpoint H-matrix are being updated and assigns a weight to the updated elements. Hence, in this project, the energy estimator assigns a weight to the x-coordinate when comparing the robot's real H-matrix to the budget-generation H-matrices in $\log_H$. This way, if the robot would experience different rotations during trajectory performance, the energy estimator keeps providing its budgets mainly based on the robot's x-axis position.

The energy estimator does not check all of the rows in $\log_H$ for every control iteration. Consider, for example, that during a control iteration it is found that the best-matching H-matrix is at row-index 50. Then, at the next control iteration only a certain range of H-matrices around that row index is checked, e.g. rows 30 to 70. This is a more efficient approach than checking every single row in $\log_H$. It is also a valid approach, because the simulated robot represents a physical system that cannot make instantaneous (discrete) jumps in pose. One would expect that, without a virtual wall, both simulations are exactly the same. However, because ROS is non real-time, simulations that are repeated in exactly the same conditions do not have the same outcome (due to time-dependencies of components). Therefore, a range of rows needs to be checked, because H-matrices in $\log_H$ are close to, but different from the H-matrices in the normal simulation.

The approach of checking H-matrices only for a certain range of rows in $\log_H$ does bring forward a new problem. Because the budget-generation and normal simulations are not exactly the same, local minima can emerge when searching for the best-matching H-matrix in $\log_H$. If the range of rows that are being checked is not large enough to cover the valley in which a local minimum lies, the energy estimator gets stuck at that local minimum. This issue has been overcome by applying a moving average (low-pass) filter over the H-matrices in $\log_H$ at the start of the normal simulation (to smooth out local minima) and by using a line-number range of (empirically determined) sufficient size. However, in case the energy estimator does get stuck, it also keeps track of how many iterations it has been at the same row index. If it has remained at the same row index for a certain percentage of the total amount of rows, it will jump forward that percentage of rows in a attempt to recover row-index detection.

E Instructions

The following corresponds to Section 4.2.6.

As described in Section 4.2.6, two types of simulation have been implemented: a normal simulation and a budget-generation simulation. They can be ran as follows:

- To run the normal simulation (including the virtual wall), execute the following[1]:
  `$ roslaunch sequence_control esca_demo.launch`
  to remove the virtual wall, add `wall:=false`
- To run the budget-generation simulation, execute the following:
  `$ roslaunch sequence_control esca_demo.launch budgetGeneration:=true`

Entering `ctrl+c` terminates a simulation.

For this serial simulation approach to work, the budget-generation simulation needs to run prior to the normal simulation (if energy budgets have not already been logged). The budget-generation simulation only needs to run once. The normal simulation can be executed as many times as desired. To obtain accurate energy budgets, there should be no developer-defined differences between both simulations other than the virtual wall (i.e. no differences in targets provided by the supervisory controller, for example). Otherwise, the energy-budget estimation based on H-matrix comparison will not function accurately.

Note: the system's C++ code has been extensively documented in Doxygen. Refer to this documentation for specific details (e.g. parameter tuning) of the system's implementation.

[1] All programming files in this project have been marked by the prefix 'esca', which stands for Energy-aware Sequence-Control Architecture. This prefix is intended to easily distinguish files related to this project among files of other projects.

F Setup

The following corresponds to Section 5.1.

The setup used for the experiments described in Section 5.1 is:

- Hardware: Lenovo ThinkPad P1 Gen 2 (20QUS), i7-9750H @ 2.60 GHz, 16 GB RAM, Windows 10.
  Note: 'best performance' mode is needed to extract proper simulation data.
- Virtual machine software: Oracle VM VirtualBox[1] v6.1.2.
  Settings:
  - Ubuntu[2] 16.04.
    Note: the simulation is not yet functional on newer Ubuntu versions.
  - 6 CPUs.
  - 11.262 GB memory.
  - 128 MB video memory.
  - 30 GB storage.
    Note: 30 GB storage has been used, but (at least) 40 GB storage is recommendable for future work.
- Middleware: ROS[3] Kinetic Kame.
  Note: this project's implementation may not be directly compatible with newer ROS versions, because the implementation has been built on the work by Lazar (2019), which contains dependencies that are not compatible with newer ROS versions.
- Simulator: Gazebo[4] 7.16.

[1] Oracle VM VirtualBox website: *https://www.virtualbox.org/*
[2] Ubuntu website: *https://ubuntu.com/*
[3] ROS website: *https://www.ros.org/*
[4] Gazebo website: *http://gazebosim.org/*

Bibliography

Alami, R., A. Albu-Schäeffer, A. Bicchi, R. Bischoff, R. Chatila, A. De Luca, A. De Santis, G. Giralt, J. Guiochet, G. Hirzinger, F. Ingrand, V. Lippiello, R. Mattone, D. Powell, S. Sen, B. Siciliano, G. Tonietti and V. Luigi (2006), Safe and dependable physical human-robot interaction in anthropic domains: State of the art and challenges, in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1–16, doi:10.1109/IROS.2006.6936985. https://doi.org/10.1109/IROS.2006.6936985

Avizienis, A., J.-C. Laprie, B. Randell and C. Landwehr (2004), Basic Concepts and Taxonomy of Dependable and Secure Computing, *IEEE Transactions on Dependable and Secure Computing*, **vol. 1**, pp. 11–33, ISSN 1545-5971, doi:10.1109/TDSC.2004.2. https://doi.org/10.1109/TDSC.2004.2

Bardaro, G., D. A. Cucci, L. Bascetta and M. Matteucci (2014), A Simulation Based Architecture for the Development of an Autonomous All Terrain Vehicle, in *Simulation, Modeling, and Programming for Autonomous Robots*, Springer International Publishing, Cham, pp. 74–85, ISBN 978-3-319-11900-7, doi:10.1007/978-3-319-11900-7_7. https://doi.org/10.1007/978-3-319-11900-7_7

Bennett, S. (1988), *Real-Time Computer Control*, Prentice-Hall International Series in Systems and Control Engineering, Prentice Hall, ISBN 978-0-13-762485-0.

Bezemer, M. M. (2013), *Cyber-physical systems software development: way of working and tool suite*, Ph.D. thesis, University of Twente, Netherlands, doi:10.3990/1.9789036518796. https://doi.org/10.3990/1.9789036518796

Bolton, W. (2015), Chapter 13 - Control Systems, in *Instrumentation and Control Systems (Second Edition)*, Newnes, pp. 281 – 302, second edition edition, ISBN 978-0-08-100613-9, doi:10.1016/B978-0-08-100613-9.00013-4. https://doi.org/10.1016/B978-0-08-100613-9.00013-4

Breedveld, P. (1985), Multibond graph elements in physical systems theory, *Journal of the Franklin Institute*, **vol. 319**, pp. 1 – 36, ISSN 0016-0032, doi:10.1016/0016-0032(85)90062-6. https://doi.org/10.1016/0016-0032(85)90062-6

Breedveld, P. C. (1982), Thermodynamic Bond Graphs and the Problem of Thermal Inertance, *Journal of the Franklin Institute*, **vol. 314**, pp. 15 – 40, ISSN 0016-0032, doi:10.1016/0016-0032(82)90050-3. https://doi.org/10.1016/0016-0032(82)90050-3

Brodskiy, Y. (2014), *Robust autonomy for interactive robots*, Ph.D. thesis, University of Twente, Netherlands, doi:10.3990/1.9789036536202. https://doi.org/10.3990/1.9789036536202

Broenink, J. F., Y. Ni and M. A. Groothuis (2010), On model-driven design of robot software using co-simulation, in *Proceedings of SIMPAR 2010 Workshops International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, TU Darmstadt, pp. 659–668, ISBN 978-3-00-032863-3. https://research.utwente.nl/en/publications/on-model-driven-design-of-robot-software-using-co-simulation

Brugali, D. and P. Scandurra (2009), Component-Based Robotic Engineering (Part I) [Tutorial], *IEEE Robotics & Automation Magazine*, **vol. 16**, pp. 84–96, ISSN 1070-9932, doi:10.1109/MRA.2009.934837. https://doi.org/10.1109/MRA.2009.934837

Bruyninckx, H., M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi and D. Brugali (2013), The BRICS Component Model: A Model-Based Development Paradigm

for Complex Robotics Software Systems, in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, Association for Computing Machinery, New York, NY, USA, SAC '13, p. 1758–1764, ISBN 9781450316569, doi:10.1145/2480362.2480693.
https://doi.org/10.1145/2480362.2480693

Carlson, J., R. Murphy and A. Nelson (2004), Follow-up Analysis of Mobile Robot Failures, *Proceedings - IEEE International Conference on Robotics and Automation*, doi:10.1109/ROBOT.2004.1302508.
https://doi.org/10.1109/ROBOT.2004.1302508

Deitel, H. M. and P. J. Deitel (2017), *C++ How to Program, Tenth Edition*, Pearson Education Limited, 10th edition, ISBN 978-1-292-15334-6.
https://deitel.com/c-plus-plus-how-to-program-10-e/

Delgado, A., C. A. Jara and F. Torres (2017), Adaptive Tactile Control for In-Hand Manipulation Tasks of Deformable Objects, *The International Journal of Advanced Manufacturing Technology*, **vol. 91**, pp. 4127–4140, ISSN 0268-3768, 1433-3015, doi:10.1007/s00170-017-0046-2.
https://doi.org/10.1007/s00170-017-0046-2

Duindam, V., A. Macchelli, S. Stramigioli and H. Bruyninckx (2009), *Modeling and Control of Complex Physical Systems: The Port-Hamiltonian Approach*, Springer, Berlin, ISBN 978-3-642-03195-3 978-3-642-03196-0, doi:10.1007/978-3-642-03196-0, oCLC: 845558767.
https://doi.org/10.1007/978-3-642-03196-0

Folkertsma, G. A. and S. Stramigioli (2017), Energy in Robotics, *Foundations and Trends® in Robotics*, **vol. 6**, pp. 140–210, ISSN 1935-8253, doi:10.1561/2300000038.
https://doi.org/10.1561/2300000038

Franken, M. (2011), *Control of haptic interaction : an energy-based approach*, Ph.D. thesis, University of Twente, Netherlands, doi:10.3990/1.9789036531894.
https://doi.org/10.3990/1.9789036531894

Greeff, G. and G. Ranjan (2004), 3 - System hierarchies and components, in *Practical E-Manufacturing and Supply Chain Management*, Newnes, Oxford, pp. 26 – 65, ISBN 978-0-7506-6272-7, doi:10.1016/B978-075066272-7/50006-3.
https://doi.org/10.1016/B978-075066272-7/50006-3

Groothuis, M. A., R. Frijns, J. Voeten and J. F. Broenink (2009), Concurrent Design of Embedded Control Software, in *Proceedings of the 3rd International Workshop on Multi-Paradigm Modeling (MPM2009)*, volume 21 of *Electronic Communications of the EASST*, European Association for the Study of Science and Technology, Netherlands, ISBN 1863-2122, doi:10.14279/tuj.eceasst.21.284.
https://doi.org/10.14279/tuj.eceasst.21.284

Groothuis, S. S., G. A. Folkertsma and S. Stramigioli (2018), A General Approach to Achieving Stability and Safe Behavior in Distributed Robotic Architectures, *Frontiers in robotics and AI*, **vol. 5**, ISSN 2296-9144, doi:10.3389/frobt.2018.00108.
https://doi.org/10.3389/frobt.2018.00108

Kawamoto, H. and Y. Sankai (2004), Power Assist Method Based on Phase Sequence Driven by Interaction between Human and Robot Suit, in *RO-MAN 2004. 13th IEEE International Workshop on Robot and Human Interactive Communication (IEEE Catalog No.04TH8759)*, IEEE, Kurashiki, Okayama, Japan, pp. 491–496, ISBN 978-0-7803-8570-2, doi:10.1109/ROMAN.2004.1374809.
https://doi.org/10.1109/ROMAN.2004.1374809

Lazar, A. (2019), *Safe use of robotic arms as input devices*, Master's thesis, University of Twente, Robotics and Mechatronics.
http://essay.utwente.nl/77908/

Lootsma, M. (2008), *Design of a Global Software Structure and Controller Framework*, Master's thesis, University of Twente, Robotics and Mechatronics.
https://essay.utwente.nl/58246/

Morales, G. A. G. (2016), *Increasing the Autonomy of the Pipe Inspection Robot PIRATE*, Master's thesis, University of Twente, Robotics and Mechatronics.
https://essay.utwente.nl/71300/

Paynter, H. M. (1961), *Analysis and Design of Engineering Systems: Class Notes for M.I.T. Course 2,751*, M.I.T. Press, Cambridge, Mass.
http://hdl.handle.net/2027/mdp.39015064874921

Pfeffer, A., T. Glück and A. Kugi (2016), Soft Landing and Disturbance Rejection for Pneumatic Drives with Partial Position Information, *IFAC-PapersOnLine*, **vol. 49**, pp. 559–566, ISSN 24058963, doi:10.1016/j.ifacol.2016.10.661.
https://doi.org/10.1016/j.ifacol.2016.10.661

Rahal, R., F. Abi-Farraj, P. R. Giordano and C. Pacchierotti (2019), Haptic Shared-Control Methods for Robotic Cutting under Nonholonomic Constraints, in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8151–8157, doi:10.1109/IROS40897.2019.8968494.
https://doi.org/10.1109/IROS40897.2019.8968494

Sommerville, I. (2016), *Software Engineering*, Pearson Education Limited, chapter 16, 10th edition, ISBN 978-0-13-394303-0.
https://dinus.ac.id/repository/docs/ajar/Sommerville-Software-Engineering-10ed.pdf

Stramigioli, S. (2001), *Modeling and IPC Control of Interactive Mechanical Systems: A Coordinate-free Approach*, number 266 in Lecture Notes in Control and Information Sciences, Springer, ISBN 1-85233-395-2, doi:10.1007/BFb0110400, editors: M. Thoma, M. Morari.
https://doi.org/10.1007/BFb0110400

Stramigioli, S. and H. Bruyninckx (2001), Geometry and Screw Theory for Robotics.
https://www.semanticscholar.org/paper/95346a7af34ea62e606c46f1268e0d3584d06c95

Tadele, T. (2014), *Human-friendly robotic manipulators: safety and performance issues in controller design*, Ph.D. thesis, University of Twente, doi:10.3990/1.9789036537841.
https://doi.org/10.3990/1.9789036537841

Willems, J. C. (1972), Dissipative dynamical systems part I: General theory, *Archive for Rational Mechanics and Analysis*, **vol. 45**, pp. 321–351, ISSN 1432-0673, doi:10.1007/BF00276493.
https://doi.org/10.1007/BF00276493