Technische Universiteit
Eindhoven
University of Technology

Control Systems Technology Department

Reviving the Hexapod

Open Space Project

R.A. Ottervanger BSc

Supervisors:
dr.ir. M.J.G. van de Molengraft

CST 2014.045

Eindhoven, April 2014

Abstract

The RoboCup project aims to have robots beat the human soccer world champion in 2050. Currently, the mid-size league depends on omni-wheels to move around on the soccer pitch. This is reasonable for a carpet-like surface or artificial turf, but on real grass, this technology is not sufficient. The University of Technology Eindhoven developed a Hexapod robot designed to walk, jump and run, aiming to fill this gap. Basic low-level software for this robot was developed in an earlier thesis, but after that thesis, the project was shut down temporarily. After more than a year of silence, this open space project is set up to revive the Hexapod and discover its possibilities.

To get the project started, first the software version was updated and problems arising from this update were solved. Then the real time loop was improved by solving an error in the communication between software components in this loop. These components were modified such that they can only be started if given physically valid parameters, to make starting them fool-proof. The safety component was rewritten such that the robot is able to recover from a safety shut-down of an actuator. Also the gravity compensation component was rewritten to correct a theoretical error. All of this made it possible to make the robot walk better than it had before. All working software was documented on the Hexapod wiki page.

Additionally, a leg tip trajectory is given to decrease impact forces with the ground while walking, but this method is not yet implemented. Finally also an initial approach is given to making the Hexapod jump. The leg tip trajectory to do this is implemented and tested, but so far without the success of making the robot jump.

Contents

Chapter 1

Introduction

1.1 Motivation and background

The RoboCup project is a project that aims to improve robots by challenging teams to compete in a robot soccer league. The goal of the project is to be able to have autonomous robots beat the human world champions in 2050. Currently, the mid-size league robots depend on omni-wheels. These wheels are designed to actively propel in the rotational direction, but also passively roll in the direction perpendicular to the wheel disk. These are however not very suitable for moving on a grass soccer pitch. The next logical step is to take the transition to legged locomotion. This is where the Hexapod comes in.

1.2 Problem statement and objective

The Hexapod robot is a robot designed to walk, hop, jump and run. It is designed such that it satisfies the requirements for a RoboCup robot in the mid-size league, aiming to take the next step towards robotic soccer on a real soccer pitch. So far it has only walked slowly, in which case it lacked stability; moreover, it has not moved at all in over a year. Therefore, the goal of this project is to revive the Hexapod, bringing it back to its old capabilities, documenting the existing software that is present at the end of the project and to explore further possibilities and limitations of the robot.

1.3 Outline

To ensure that the information recovered during this project is not lost again, this report aims to document the software that is available at the end of this project. To be able to give an explanation of the software, first an overview of the hardware is given in chapter 2. Then the way the system is modeled is explained in chapter 3. Finally, in chapter 4 the software that runs on the Hexapod is explained. Because specific development was done regarding jumping, this subject is elaborated upon further in chapter 5.

Chapter 2

Hardware

This chapter describes the main hardware components that are necessary for the understanding and design of software for the Hexapod. For more detailed information on the hardware design, refer to [8]. For more hands-on instructions on start-up, refer to the Hexapod's Wiki page [2].

2.1 Power supply

The power to the robot can be supplied by one of the 8 cell lithium polymer battery packs that can be mounted on the bottom of the robot. For testing purposes however, the more practical solution to powering the robot is to use a laboratory power supply. The power supply should be set to 24 V and the current should not be limited to less than 10 A because the motors can draw high peak currents.

2.2 Legs, actuators and transmission

The legs consist of three links. From body to tip, these are called the coxa, femur and tibia. The actuators on the Hexapod are Maxon EC 45 flat brushless 30 Watt motors. There is one actuator for each joint, which gives the robot 18 internal degrees of freedom, offering a large freedom of movement compared to robots with, for instance, coupled joints. The actuators are connected to torsional springs through a GS38A Maxon 60:1 gear head, so the system can be regarded as one with Series Elastic Actuation. These torsional springs offer the possibility to store elastic energy when a force is applied to the leg tip. A drawback of the springs is that they also imply that the control bandwidth of the joints is rather low. Furthermore, the actuator torques for the femur and tibia links are transferred from the end of the torsional spring to the joint through steel cables. The cable between the coxa and the femur has the tendency to break, so there is a spare leg that can replace a broken leg when necessary.

2.3 Sensors

The legs are equipped with magnetic single turn absolute encoders. This means that they do not need calibration when the robot is started. They need to be calibrated only after changing a leg or other repair work that may have changed the zero point of the encoder itself, such as replacing a broken steel cable. An example of an absolute encoder disk is shown in figure 2.1. Each joint is equipped with two encoders, one of which is placed in the actuator and the other at the joint itself.

The built-in CHR-6d IMU (Inertial Measurement Unit) measures linear acceleration along three axes, rotation rates about three axes and pitch and roll angles.

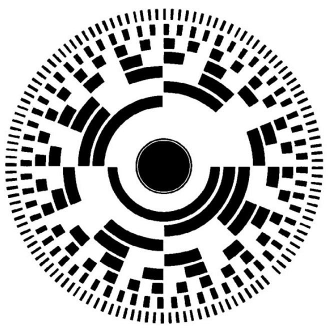


Figure 2.1: *Schematic visualization of an absolute encoder disk. Signals may be read magnetically or optically.*

2.4 Data acquisition

The real time data acquisition is performed by two Beckhoff EtherCAT FB111-0142 piggyback controller boards. The first one is connected to the motors and encoders, the other is connected to the IMU and the power supply, which may be a battery or an external power supply. These modules can be connected to the on board PC, but as this PC is currently not operational, the robot is controlled by an off-board laptop PC.

Chapter 3

System model

The software for the Hexapod is based on a simplification of reality. In this chapter this model of the robot system is explained. To do this, first the used reference frames are illustrated in section one. In the second section, the dynamic model of the robot body is explained.

3.1 On the used reference frames

In order to elaborate on the theory of the Hexapod, several frames of reference are needed. The robot moves its body freely space with respect to a world frame, it has six legs that each have their own reference frame in Cartesian space, these frames can be expressed with respect to the body fixed frame and additionally, there is a direction oriented reference frame. The relationships between these frames and their meanings are explained in this section.

3.1.1 World frame

The world frame such that

$$\boldsymbol{X} = \mathcal{W}\mathbf{X}^W = \left[\vec{e}_1^{\,W}, \vec{e}_2^{\,W}, \vec{e}_3^{\,W}\right] [X, Y, Z]^T \tag{3.1}$$

is the inertial frame. It has its Z-axis upwards from the ground, and the X and Y axes oriented such that they form a right handed frame. The most obvious choice for the X axis is the initial body x axis. The world frame is also the frame the body height and pitch and roll angles are expressed in. Also odometry information would typically be expressed in this frame.

3.1.2 Body fixed frame

As the name implies, the body fixed frame such that

$$\boldsymbol{x} = \mathcal{B}\mathbf{x}^B = \left[\vec{e}_1^{\,B}, \vec{e}_2^{\,B}, \vec{e}_3^{\,B}\right] [x, y, z]^T \tag{3.2}$$

is attached to the robot body. It has the z axis pointing up, the x axis towards leg 1 and the y axis such that the right handed frame is completed. It translates and rotates with the robot in all directions. This frame spans what is commonly referred to as the operational space. The tip coordinates of the legs are represented in this base before they are converted to joint space by the inverse kinematics. The body fixed frame and world frame are displayed in figure 3.1

3.1.3 Direction-of-movement fixed frame

Direction based frame such that

$$\boldsymbol{\xi} = \mathcal{D}\boldsymbol{\xi}^D = \left[\vec{e}_1^{\,D}, \vec{e}_2^{\,D}, \vec{e}_3^{\,D}\right] [\xi_x, \xi_y, \xi_z]^T \tag{3.3}$$

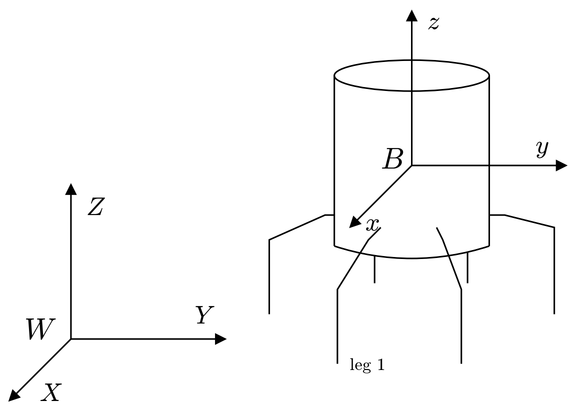


Figure 3.1: *Relation of the body fixed frame and the world frame*

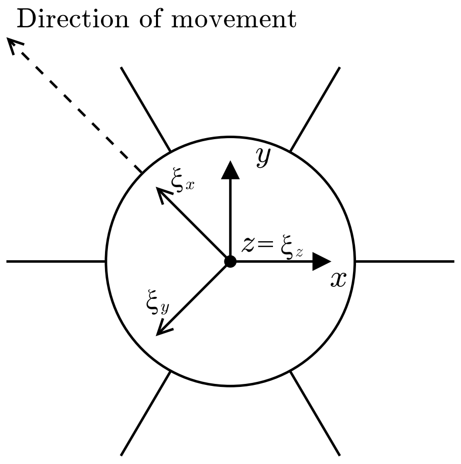


Figure 3.2: *Relation of the body fixed frame and the direction-of-movement fixed frame*

is oriented such that its ξ_z axis coincides with the (body) z axis and the ξ_x axis is always in the horizontal direction of movement. The reference trajectories for the leg tips are defined in this reference frame to be independent of the direction of movement. For clarity, this is illustrated in figure 3.2.

3.1.4 Leg reference frame

Each leg has its own reference frame, where

$$\boldsymbol{x} = \mathcal{L}\mathbf{x}^L = \left[\vec{e}_1^{\,L}, \vec{e}_2^{\,L}, \vec{e}_3^{\,L}\right] [\mathrm{x}, \mathrm{y}, \mathrm{z}]^T \tag{3.4}$$

in which the leg Jacobian is defined. The z-axis of this system is along the axis of the first joint, parallel to the body z axis, the x-axis is pointed outward and the y-axis completes the right-handed frame. The origin of this system is located at the z position corresponding to the height of the second joint.

3.1.5 Joint space

Finally, the tip position is controlled in joint space, spanned by

$$\boldsymbol{q} = \mathcal{J}\mathbf{q}^J = \left[\vec{e}_1^{\,J}, \vec{e}_2^{\,J}, \vec{e}_3^{\,J}\right] [\theta_1, \theta_2, \vartheta_3]^T. \tag{3.5}$$

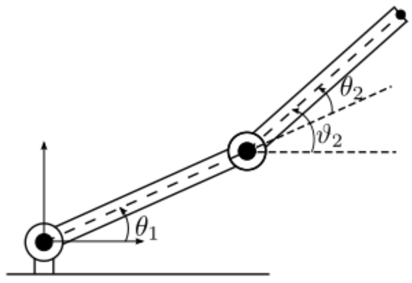


Figure 3.3: *An example of the relation between relative angle θ and absolute angle ϑ.*

Note that the final element in the joint space vector is

$$\vartheta_3 = \theta_3 + \theta_2, \tag{3.6}$$

such that it is the angle as used in the Denavit-Hartenberg convention [6]. This is not entirely trivial since the joint angle that is measured by the actuator encoder is θ_3 as this joint is physically decoupled from joint 2. The distinction is illustrated in figure 3.3.

3.2 Body dynamics

A large part of this section is adopted directly from [8]. However, a fundamental improvement was made to his elaboration, being that the correct vector bases are now taken into account in the matrix calculations. The forces and moments acting on the robot in normal operation are only the reaction forces from the ground on the leg tips. As the leg tips are unable to transfer moments, only forces are applied to the leg tips. The total force and moment vectors applied to the robot's body by the legs can thus be expressed as

$$\boldsymbol{F} = \sum_{i=1}^{p} \boldsymbol{f}_i, \tag{3.7}$$

$$\boldsymbol{M} = \sum_{i=1}^{p} \boldsymbol{r}_i \times \boldsymbol{f}_i, \tag{3.8}$$

where i is the leg index, p the number of legs on the floor, $\boldsymbol{r}_i$ the position vector of the tip of leg i and $\boldsymbol{f}_i$ the force vector acting on that leg tip. The force vector can be expressed in matrix form as

$$\mathbf{F} = \mathbf{J_F}\mathbf{f} \tag{3.9a}$$

$$= \begin{bmatrix} \mathbf{I}_3 & \cdots & \mathbf{I}_3 \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_p \end{bmatrix} \tag{3.9b}$$

where $\mathbf{J_F} \in \mathbb{R}^{3\times 3p}$ and $\mathbf{f} \in \mathbb{R}^{3p}$. And the moment vector can be rewritten in a similar way as

$$\mathbf{M} = \mathbf{J_M}\mathbf{f} \tag{3.10a}$$

$$= \begin{bmatrix} 0 & -r_{z,1} & r_{y,1} & \cdots & 0 & -r_{z,p} & r_{y,p} \\ r_{z,1} & 0 & -r_{x,1} & \cdots & r_{z,p} & 0 & -r_{x,p} \\ -r_{y,1} & r_{x,1} & 0 & \cdots & -r_{y,p} & r_{x,p} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_p \end{bmatrix} \tag{3.10b}$$

where $\mathbf{J_M} \in \mathbb{R}^{3\times 3p}$ and $r_{x,i}$, $r_{y,i}$ and $r_{z,i}$ are components of $\boldsymbol{r}_i$ expressed in the body fixed frame. Note that in equations 3.9 and 3.10, no longer italic fonts are used for $\mathbf{F}$ and $\mathbf{M}$, indicating that these are matrices expressed with respect to a base; more specifically, the body fixed vector base $\mathcal{B}$. Superscripts are omitted for readability. These two matrix equations can be combined to

$$\mathbf{W} = \mathbf{J_{FM}}\mathbf{f}, \tag{3.11}$$

with

$$\mathbf{J_{FM}} = \begin{bmatrix} \mathbf{J_F}^T & \mathbf{J_M}^T \end{bmatrix}^T \in \mathbb{R}^{6\times 3p},$$
$$\mathbf{W} = \begin{bmatrix} \mathbf{F}^T & \mathbf{M}^T \end{bmatrix}^T \in \mathbb{R}^{6}$$

and $\mathbf{W}$ denoting the wrench. Using this equation, the required tip force distribution for equilibrium can be calculated from the wrench expressed in the body fixed frame. As the wrench is known in inertial space, i.e. the world reference frame, it should first be converted to the body fixed frame before it can be applied to equation 3.11. This can be done using the rotation matrix $\mathbf{R}_W^B$ relating the world frame to the body fixed frame as

$$x^B = \mathbf{R}_W^B X^W. \tag{3.12}$$

As equation 3.11 is under-determined, it has multiple solutions. It can however be solved minimizing $||\boldsymbol{f}||$ using the pseudo inverse of $\mathbf{J_{FM}}$, which is numerically efficiently obtained using the singular value decomposition of $\mathbf{J_{FM}}$ [3]
Finally, the required joint torques for a leg can be found by premultiplying the tip force vector of the respective leg with the transpose of its Jacobian. For the derivation of the Jacobian, refer to [8]. As this Jacobian is defined with respect to the leg root, the force vector needs an additional rotation before premultiplication with the Jacobian. This results in

$$\boldsymbol{\tau}_i^J = \mathbf{J}^T \mathbf{R}_{l,k}^B \mathbf{f}_i^B \tag{3.13}$$

giving the required joint torques for leg i.

Chapter 4

Software

This chapter aims to explain the software that is used for the Hexapod. A large part of this software was already written by [8]. However, many improvements were made varying from small fixes to corrections of fundamental flaws. For information about the important improvements, especially note the sections about the framework, the safety component and the gravity compensation.

4.1 Framework

The software that is used for the Hexapod is based on Orocos. Orocos stands for Open RObot COntrol Software. This is an open source framework that lets the user create *components* which can communicate with each other using their *ports*. The power of this framework lies in RTT (Real Time Toolkit), which makes it possible to run these components (almost) in real time, yielding low latency and predictable behaviour. This is especially useful when controlling hardware. For a detailed explanation about Orocos and its usage, refer to the Orocos Component Builder's Manual [4]

The next software level will be based on ROS, the Robot Operating System. This is a framework based on *nodes* communicating through *topics*, over which *messages* are sent. This structure, which is similar to that of Orocos, offers the possibility to separate software tasks and divide them into different nodes. ROS is also used for the TechUnited care robot Amigo and will be used for the TechUnited soccer robots (TURTLEs) in the near future, the latter making it an especially good choice, making sure that the software for the Hexapod is compatible with the higher level software of the existing soccer robots.

The ROS distribution that was originally used for the Hexapod is ROS Electric, whereas the used distribution by TechUnited, the University of Technology Eindhoven RoboCup team, at the start of this project was ROS Fuerte. To be up to date with the rest of the team, migration was performed to Fuerte. Up to this point, ROS is only used for interfacing with RViz, a visualization interface for the robot's perception of itself and its environment. However, the streams that pass data from Orocos to ROS were not working, giving the error "`Need a transport for creating streams`". This problem was solved by giving appropriate sizes to the streams. The joint states and IMU data are sent to RViz, making it possible to visualize the robot with its actual current leg configuration and its pitch and roll angles.

The next step in usage of ROS, is to use it to generate a *Twist* message, and send it to the Orocos software. A Twist message is a message that contains two vectors: one of linear velocities and one of angular velocities. The Orocos software should be able to translate such a message to a gait pattern and control the leg joints such that the robot follows this Twist message.

As stated by [8], a URDF (Unified Robot Description) model was also built for the Hexapod, making it possible to use Gazebo, a robot physics simulation program, to simulate the robot's behaviour. At the end of this project this simulator can be started and the model can be spawned, but the interfacing with Orocos is not yet recovered.

4.2 Architecture

In this section, the general architecture of the real time software is explained. This architecture is based on a feedback loop with on line reference generation in Cartesian space and was set up by [8]. A schematic overview of the control loop is shown in figure 4.1. In this figure, observe the four driver components on the right hand side (including SOEM). Together, these form the hardware interface. Also observe the reference generation on the left hand side. In between these parts, there is a standard feedback loop: the error is calculated and sent through several control filters before it is sent to the hardware interface (in this case through a safety component). In section 4.3, the separate components are explained more into detail.

Orocos components can be configured to run at a fixed frequency. This means that the *updateHook* is executed every time step, reading data from its input ports, executing calculations on them and passing its output data to the next components in line. Originally, the Hexapod's Orocos components were configured in this manner. This explains the delay of approximately 9 ms found by [8] in the joint control loop. This however results in a delay of one time step for each component in the loop, as the next component in line only responds to the new data when it has reached the next time step. This means that with a loop of, say, ten components and a time step of 0.001 s, the delay is already 0.01 s, increasing linearly with the number of components in the loop. Also, the frequency of the components is never actually deterministic, which means that the time instances at which data is transferred from one component to another is non-deterministic, resulting in unpredictable behaviour.

To realize faster and more predictable behaviour, regular input ports are now replaced by *EventPorts*. In this concept, one component runs at a fixed frequency, running its updateHook at this frequency. Another component, connected to the output port of this first component using an EventPort, is triggered by this first component, also executing its updateHook. A third component can then be triggered by this second component and so on. This means that the next component can get started processing data the moment it receives data from upstream, resulting in a much faster loop.

In this project, the timing strategy of the Orocos components was changed from the periodic to

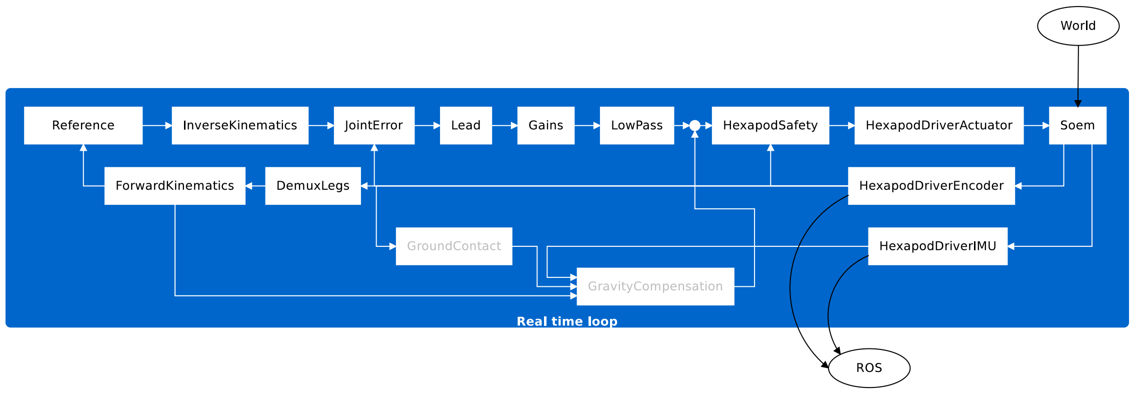


Figure 4.1: *Schematic overview of the Hexapod control loop. Components with gray text are implemented but not in use because of inaccuracy.*

event-triggered. The SOEM component is used as the periodic component triggering the components in the loop. The reference generation, if not triggered by the forward kinematics, is also set to this fixed frequency. Before this change, the robot did not move smoothly and while standing, it made a stuttering noise, sagging slightly with every jolt. The implementation of EventPorts removed the stuttering noise and moreover, it resolved the sagging and resulted in smoother movements.

Orocos components also have *properties*; these can be used as parameters in the component program. Properties are used for controller components to set the control parameters, but also for robot characteristics such as weight, dimensions, encoder offsets and joint limits. In the previously existing software, components could be started without setting their properties. This would result in software crashes or worse, unpredictable behaviour of the hardware. In this project, all used components that showed this unsafety were fixed. The implemented solution was to pre-assign the properties with a physically meaningless value and adding an assertion to check the physical validity of the properties. This assertion is performed in the *startHook*, which is the first method to be executed when a component is started and which is performed only once. This implementation ensures that the component can only be started when the properties are properly set, which makes starting components fool-proof.

4.3 Components

This section shows the workings of the separate Orocos components that are written for control of the Hexapod. Also it explains how these components are deployed and how they interact with each other.

4.3.1 Launching and deployment

Orocos components can be started using Orocos' *TaskBrowser*. This is a command line application that prompts for commands to pass to the *deployer*. Orocos components are deployed as a plugin to the deployer. For more into detail information on deployment, refer to the documentation on the Deployment component [5]. Especially when starting larger numbers of components, using the TaskBrowser manually to start components one by one becomes more time consuming. Therefore, it is possible to list the necessary commands for the deployer in a script (.ops file). These scripts can call each other, and can in turn be called by ROS' launch files. Keeping in mind the goal of running software from ROS, it is convenient to start a deployment script using a launch file that can be started using ROS. At this point, several .ops scripts call other scripts, which needs hard coded paths. A script was written to find these paths and change them to the user path. This is not very user-friendly, so in time, it may be best to either combine these scripts into larger scripts, or launch these scripts from a ROS launch file, which is capable of using ROS' environment variables.

4.3.2 SOEM

The Hexapod's Orocos software is built around a present driver for the EtherCAT boards, called SOEM, or Simple Open EtherCAT Master. For this master, a wrapper exists, making integration with Orocos possible. Also a driver exists for the Hexapod enabling very basic communication between Orocos components and the robot. This driver is depicted with Soem in figure 4.1. Other components are constructed to interface with the basic interface of the Hexapod's SOEM driver.

4.3.3 Encoder driver

The encoder signals are read from the SOEM component by the encoder driver. This component converts the counts from the encoders to angles in radians. For that it needs a zero position, which is read from a properties file during deployment. The encoder driver takes into account the number of revolutions, such that when the encoder passes its start/end point (its count value goes from very high to very low in one time step), 2π is added to the output angle and vice versa. The encoder driver also converts the absolute angle θ_3 that is measured by the actuator encoders to the relative angle ϑ_3; the joint encoders already give the relative angle. When a leg is replaced or repaired, often the joint encoders are not placed exactly as they were before, so the encoders need to be recalibrated. This can also be done using this component and the hardware_calibration.ops file.

4.3.4 Ground contact

The ground contact component is designed to determine if a leg is touching the ground or not. It utilizes the torsional springs that connect the actuators to the joints. If a force is applied to a leg, this is converted to joint torques and consequently, torques in the torsional springs. These torques lead to elastic deformation, which can be calculated from the difference between the joint angle and the actuator angle. If this deformation passes a certain threshold, the leg is assumed to be touching the ground.

There are some issues with the ground contact component. It seems that ground contact is frequently registered falsely positive. This may be caused by torques to accelerate links, also generating a torsion of the springs. Another possible cause is backlash in the gear houses and joints. Whatever the cause, if the output of the ground contact component is used as input for compensation of the weight of the robot (see subsection 4.3.12), this can result in unexpected or even unstable behaviour, especially when switching between support and transfer phase often.

4.3.5 IMU driver

The IMU driver receives its input from SOEM. It translates the raw IMU data to pitch, roll, gyro and acceleration data, removing offsets where necessary and scaling them to SI units.

4.3.6 Actuator driver

The actuator driver recieves SI units from the safety component and converts these to hardware-compatible units. Also it is equipped with a saturation to limit the demanded motor currents.

4.3.7 Safety

The safety component is designed to guard the geometric safety of the robot. Originally, this component turned the tibia actuator off when ϑ_3 went outside of its workspace. On the one hand, this is very liberal, as there are no constraints on the coxa and femur joints, although they do exist in practice. On the other hand, this safety measure is very limiting as the robot is unable to autonomously recover from such a case.

To improve this, the safety component is rewritten to ensure that an actuator is turned off when its corresponding joint limit is reached and the input signal is still towards the joint limit. When the control signal is turned away from the joint limit, back into the workspace, the actuator is turned on again. This way, the robot can recover from the error and proceed with its task. Also the other joints are now limited such that the links do not damage the body or themselves. Another possible improvement is a dynamic safety measure to ensure that the legs do not hit each other while walking or jumping.

4.3.8 Control filters

The control filters that are used for the Hexapod are part of the generic RTT components from the Systems and Control library [1]. Three filters are used: a lead filter, a low pass filter and a

Table 4.1: *Controller parameters for joint control*

	Joint 1	Joint 2	Joint 3
Lead zero [Hz]	1.7	1.7	1.7
Lead pole [Hz]	15.0	15.0	15.0
Low pass pole [Hz]	25.0	25.0	25.0
Gain [-]	-10.0	5.0	-5.0

gain, with parameters based on [8] and shown in table 4.1.

4.3.9 Joint error

The joint error component calculates the error in the joint positions from the difference between the joint encoder angle and the reference joint angle. It is triggered by the input port on which the encoder angle is received to guard the real-timeness of the loop.

4.3.10 Kinematics

There are two kintematics components. One for the forward kinematics (FK), and one for the inverse kinematics (IK). Both depend on the robot model, which is defined in a URDF-file like the Gazebo model, and on KDL (the Kinematics and Dynamics Library of the Orocos project). The forward kinematics component calculates the tip position from joint angles, while the inverse kinematics determine feasible joint angles for given leg tip positions. The latter may get input that makes it impossible to find a feasible solution. In that case, it does not send any joint reference position at all.

Each of these components is implemented in a distributed manner, so each leg has its own instance of the FK and the IK component. For now, this approach is adopted from [8], but the quality of the choice is not assessed in this project.

4.3.11 Reference generation

Several reference generation components are available, among which HexapodPosition, Hexapod-ContinuousGait and HexapodJump. These are bundled in the hexapod_gaits package. Reference generation components written by [8] are generally also implemented in a distributed way. The quality of this approach is assessed for the continuous gait component by making the robot walk. In this experiment it appears that the legs are not synchronized, which results in an irregular gait. This may be improved by using a centralized reference generation or by implementing a central clock to synchronize the components.

The HexapodPosition component uses the function `setTipPosition` to create a reference trajectory from the current position to the setpoint. This trajectory is linear in both time and in Cartesian space. This method may be improved by creating a trajectory of third order, or linear with parabolic blends [6].

The HexapodContinuousGait component creates a 2 dimensional reference trajectory for a walking gait. To do this, it uses a variable phase, which can be set differently for each leg. This way, several different gaits are possible, such as tripod, quadruped or quint The reference trajectory is defined in the ξ_x, ξ_z space (in the direction based frame). The phase here is defined as a fraction of the full cycle, so

$$\phi \in [0, 1). \tag{4.1}$$

The reference is then generated as a function of the time in the cycle

$$\tilde{t} \in [0, t_c), \tag{4.2}$$

where t_c is the cycle time. The implemented 2D reference trajectory in the continuous gait component can be expressed as

$$\xi_x = \begin{cases} -\frac{R}{\beta t_c}\tilde{t} + \frac{R}{2} & \text{if } \tilde{t} \le \beta, \\ \frac{R}{2}\cos\left(\frac{\pi}{t_c(1-\beta)}\right)(\tilde{t} - t_c\beta) & \text{else;} \end{cases} \tag{4.3a}$$

$$\xi_z = \begin{cases} H & \text{if } \tilde{t} \le \beta, \\ C\sin\left(\frac{\pi}{t_c(1-\beta)}\right)(\tilde{t} - t_c\beta) & \text{else.} \end{cases} \tag{4.3b}$$

Here, R is the stride, or step size, H is the body height, C is the foot clearance and β is the duty factor, which is the fraction of the cycle time that the foot is on the ground. This trajectory is plotted in Matlab and can be found in figure 4.2. Finally, this trajectory is converted from the direction based frame to the body fixed frame using

$$x = \xi_x \cos\gamma + r\cos\alpha_i, \tag{4.4a}$$

$$y = \xi_x \sin\gamma + r\sin\alpha_i, \tag{4.4b}$$

$$z = \xi_z. \tag{4.4c}$$

Here, γ is the angle between the body x-axis and the walking direction; r is the gait radius, which is the perpendicular distance from the body z-axis and the leg tip; and α_i is the angle between the x-axis and the line from the body z-axis to the leg z-axis of the respective leg, with i the leg index.

Observe that the position profile has a sudden change of slope at $\tilde{t} = \beta t_c$ and at $\tilde{t} = 0$, where the state changes either from support to transfer or vice versa. This means that the velocity profile is not continuous and the acceleration of the reference trajectory equals infinity or minus infinity at this point. To solve this, another reference may be implemented based on the reference trajectory designed by [9]:

$$\xi_x = \begin{cases} -\frac{R}{\beta t_c}\tilde{t} + \frac{R}{2} & \text{if } \tilde{t} \le \beta, \\ a\left(\frac{1}{5}\tilde{t}^5 - \frac{1}{2}(\beta+1)\tilde{t}^4 + \frac{1}{3}(\beta^2+4\beta+1)\tilde{t}^3 - \beta(\beta+1)\tilde{t}^2 + \beta^2\tilde{t}\right) - \frac{R}{\beta}\tilde{t} + C_2 & \text{else;} \end{cases} \tag{4.5a}$$

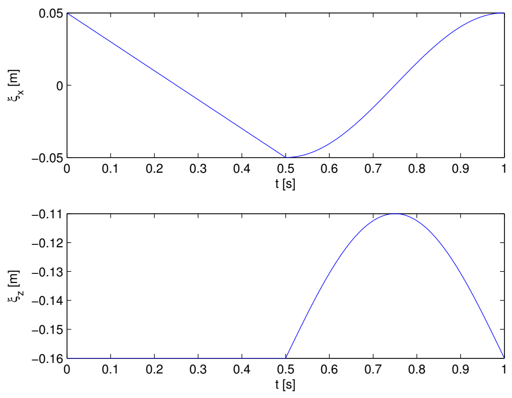
$$\xi_z = \begin{cases} H & \text{if } \tilde{t} \le \beta, \\ H + \frac{C}{2}\left(1 - \cos\left(2\pi\frac{(\tilde{t}-\beta)}{(1-\beta)}\right)\right) & \text{else;} \end{cases} \tag{4.5b}$$

where

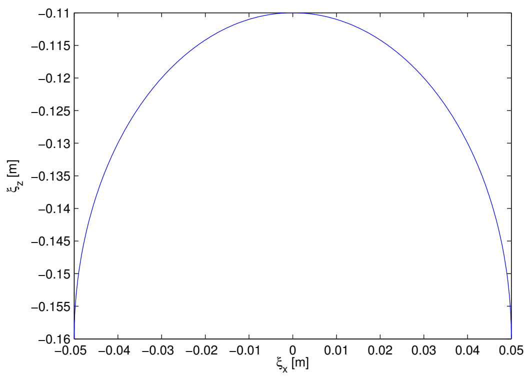
$$a = -60\frac{R}{2\beta(\beta-1)^5},$$

$$C_2 = R\frac{\beta^5 - 3\beta^4 + 10\beta^2 + 5\beta - 1}{2(\beta-1)^5}.$$

This reference trajectory is designed such that the velocity profile is continuous and the tip matches the ground speed at the point where it touches the ground. This reduces the impact force, which may make the Hexapod walk smoother. The trajectory is plotted in Matlab, which can be seen in figure 4.3 and it is ready to be implemented on the Hexapod.

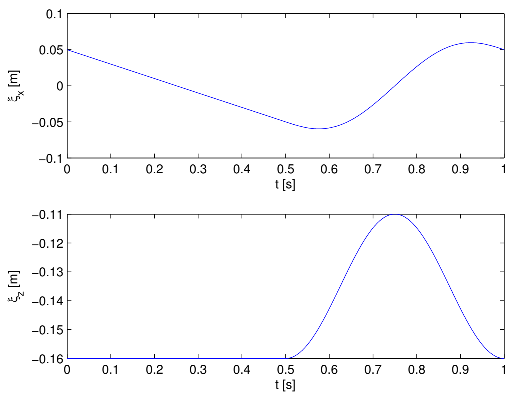


(a) Time trajectory in ξ_x and ξ_z

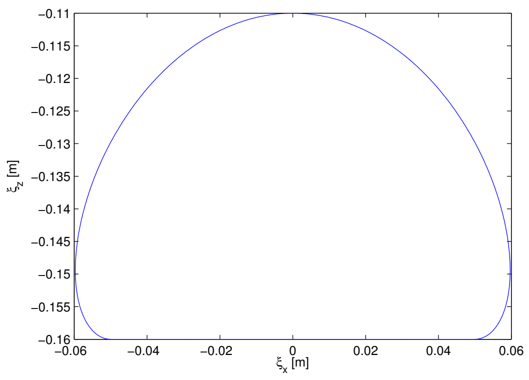


(b) Spatial trajectory: ξ_z vs ξ_x

Figure 4.2: *Reference trajectory as implemented by [8] with a step size of* 0.1 m*, a body height of* 0.16 m*, a foot clearance of* 0.05 m*, a cycle time of* 1 s *and a duty factor of* 0.5



(a) Time trajectory in ξ_x and ξ_z



(b) Spatial trajectory: ξ_z vs ξ_x

Figure 4.3: *Reference trajectory as designed by [9] with a step size of* 0.1 m*, a body height of* 0.16 m*, a foot clearance of* 0.05 m*, a cycle time of* 1 s *and a duty factor of* 0.5

4.3.12 Gravity compensation

The implementation of the gravity compensation was corrected accordingly to the fundamental improvement explained in 3.2. The component uses information from the ground contact component to determine which legs are on the ground. It only sends a nonzero force if there are at least three legs on the ground. Then the necessary vectors of forces and moments F and M are calculated in the world frame, so far this is only the gravitational force vector $\mathbf{g}^W = [0, 0, gm]^T$, where $g = -9.81$. Using the IMU data, these vectors are converted to the body fixed frame. The component then uses the current tip positions from the FK to calculate $\mathbf{J_F M}$ as in 3.11, and a distribution of leg tip forces to compensate for the gravitational force acting on the robot body.

This component may also be used to calculate tip forces needed for acceleration and deceleration during walking or jumping. It could use the acceleration data provided by the reference generation component to calculate the necessary forces on the body using Newton's second law. These forces can then be added to the gravitational force and also converted to tip forces.

4.3.13 Jacobian

The Jacobian component receives the tip positions from the FK to calculate the Jacobian matrix in the leg frame. Then it uses the calculated optimal tip forces from the gravity compensation component to calculate joint torques as explained in equation 3.13.

Chapter 5

Hopping and jumping

As stated in the introduction, the Hexapod is designed as a hopping, jumping and eventually running robot. To achieve this, the idea is to store energy in the torsional springs between the actuators and the joints. The maximum amount of energy is stored in the springs when it is excited at the resonance frequency. Therefore, this frequency is first determined. Then, using this information, a trajectory is planned for the center of mass (CoM). Finally, this CoM trajectory is converted to a leg trajectory. As a hopping Hexapod is a new phenomena, this chapter aims to highlight the progress that was made on this subject.

5.1 Resonance frequency

To get the Hexapod off the ground at once, the tip speed of the legs is not enough. Energy needs to be stored in the torsional springs of the Hexapod legs to gradually make higher jumps. To store and release the maximum amount of energy, the resonance frequency is utilized. To do this, the reference frequency needs to be found. Let us first approach the Hexapod as a single degree of freedom (DoF) mass-spring system. For such a system, the resonance frequency typically follows from

$$f_r = \sqrt{\frac{c}{m}}, \tag{5.1}$$

where c is the spring constant and m is the mass. On the Hexapod, support of the legs provides the spring stiffness. If each of the legs is modeled as a spring, the robot's body is supported by k parallel springs. The total spring stiffness of parallel springs is the sum of the stiffnesses of the springs:

$$c_{total} = \sum_{i=1}^{k} c_i, \tag{5.2}$$

where k is the number of springs. Substituting 5.2 into 5.1 results in

$$f_{r,k} = \sqrt{\frac{\sum_{i=1}^{k} c_i}{m}}, \tag{5.3}$$

with k the number of feet on the ground. If we now assume the spring stiffnesses of the legs to be equal to one another, the resonance frequency of the robot should generally equal

$$f_{r,k} = \sqrt{\frac{kc_1}{m}}, \tag{5.4}$$

where c_1 is the stiffness of one leg.

5.1.1 Experiment setup

To obtain the resonance frequency of the robot standing on six legs, an experiment is performed on the robot. In this experiment the robot's leg tips are set to a reference position

$$\boldsymbol{r}^l = (0.26, 0.26, -0.16)^T, \tag{5.5}$$

where $\boldsymbol{r}^l$ is the leg tip vector in the leg frame. Furthermore, the robot is put on the ground, standing on its leg tips and an external impulse function is applied to the robot by giving it a slight, but sudden push on the top, exciting it in its vertical mode.

The robot is expected to attain a damped vertical oscillation in its natural frequency. This frequency is recorded by reporting data from the IMU. Since no vertical position data is available, the acceleration data is used. As the response is expected to be a damped sinusoid, and the second derivative of a sinusoid is a sinusoid with the same frequency (but different phase and amplitude), the acceleration data should contain the same frequencies as the position data. The vertical acceleration data from the IMU driver is reported to a data file to be processed with Matlab.

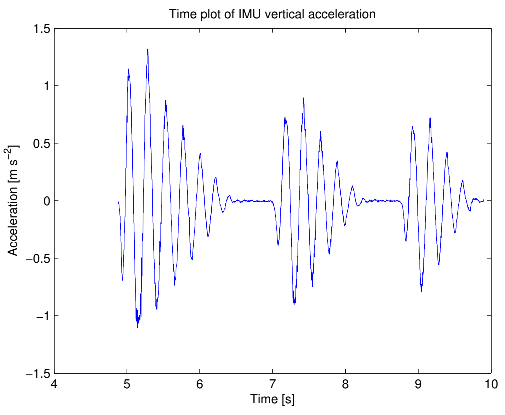
5.1.2 Results

In Matlab, both a time plot and a single sided amplitude spectrum of the measurement are constructed, which can be found in figures 5.1. The Matlab function that was written to make these figures can be found in Appendix A. As can be seen in figure 5.1a, three impulses were applied to the robot to obtain a longer measurement. Data before the first impulse and after the last visible vibrations, which are approximately zeros, are removed to avoid bucketing.

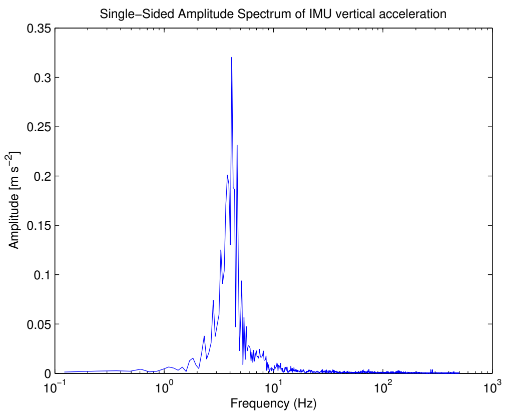
Both the time plot and the frequency spectrum show a clear presence of a 4.0 Hz signal for the experiment with six legs on the ground. This is consistent with visible and audible observations of the robot during the measurement. Therefore it is concluded that the resonance frequency of the Hexapod standing on all six legs is $f_{r,6} = 4.0$ Hz.

After this experiment it would be interesting to also do a measurement on the robot standing on only three or four legs to verify whether equation 5.4 holds or not. This may be of interest in a later stage, when one may want to influence the resonance frequency of the robot. This experiment is not yet performed because of the time limitation and a broken leg cable.

It should be noted that the experiment is done using tip position control, implying that the controller stiffness has an influence on the results. This means that when the controller is changed, or parameters are tuned differently, the experiment should be repeated, possibly resulting in a significantly different resonance frequency. Also the tip position and leg configuration may have influence on the resonance frequency.



(a) Time plot



(b) Frequency spectrum

Figure 5.1: *Results of a resonance frequency measurement of the Hexapod standing on 6 legs*

5.2 Trajectory generation

For the generation of the center of mass trajectory for the support phase, part of the approach of [7] was adopted. In this article, a CoM trajectory is generated for the support phase of a biped moving in the sagittal plane. Using that trajectory, joint trajectories are calculated. The flight phase trajectory is then generated using a quintic polynomial method.

5.2.1 Support phase

Part of the method in [7] can be applied directly to the Hexapod, using the results for the X and Z position of the CoM in the world frame

$$X_{CoM}(\hat{t}) = (X_0 - X_{zmp}) \cosh \omega\hat{t} + \frac{\dot{X}_0}{\omega} \sinh \omega\hat{t} + X_{zmp}, \tag{5.6}$$

and

$$Z_{CoM}(\hat{t}) = \left(Z_0 - \frac{g}{\omega^2}\right) \cosh \omega\hat{t} + \frac{\dot{Z}_0}{\omega} \sinh \omega\hat{t} + \frac{g}{\omega^2}. \tag{5.7}$$

Here the X position of the zero moment point is

$$X_{zmp} = \frac{X_{CoM,0} + X_{CoM,d}}{2}, \tag{5.8}$$

with the horizontal starting position X_0 and the horizontal end position $X_d = v_m T_s + X_0$, the mean velocity multiplied by the support time. if the acceleration strategy is chosen such that the horizontal velocity profile is symmetric.

As the Hexapod has a large base when it has six legs on the ground, and considering only vertical hopping, this trajectory can be used for a primitive proof of concept. If we assume that the leg tips will remain on the ground in the support phase, without slippage, and the robot body stays upright, the system only has one degree of freedom. The trajectory of the CoM with respect to the world frame can then easily be converted to leg tip trajectories in the direction oriented reference frame by inverting it and giving it an offset in the z direction.

$$\xi_{x,CoM}(\hat{t}) = -(\xi_{x,0} - \xi_{x,zmp}) \cosh \omega\hat{t} + \frac{\dot{\xi}_{x,0}}{\omega} \sinh \omega\hat{t} + \xi_{x,zmp}, \tag{5.9}$$

and

$$\xi_{z,CoM}(\hat{t}) = \xi_{z,0} + 1 - \left(1 - \frac{g}{\omega^2}\right) \cosh \omega\hat{t} + \frac{\dot{\xi}_{z,0}}{\omega} \sinh \omega\hat{t} + \frac{g}{\omega^2}. \tag{5.10}$$

This trajectory is to be followed in the support phase of every jump cycle. It is shown in the white parts of figure 5.2a, which corresponds to the part of the curve with positive ξ_z in 5.2b.

5.2.2 Flight phase

The tips now only need to be returned to their original position and velocity (and preferably acceleration) during the flight phase to enter the next support phase. This may be done using a quintic polynomial trajectory, as explained in [6]. This is done in Cartesian space in both the ξ_x and the ξ_z directions.

$$\xi_{x,f}(\hat{t}) = a_0 + a_1\hat{t} + a_2\hat{t}^2 + a_3\hat{t}^3 + a_4\hat{t}^4 + a_5\hat{t}^5, \tag{5.11}$$

$$\xi_{z,f}(\hat{t}) = b_0 + b_1\hat{t} + b_2\hat{t}^2 + b_3\hat{t}^3 + b_4\hat{t}^4 + b_5\hat{t}^5, \tag{5.12}$$

of which the coefficients can be determined equating the first and second derivatives to the boundary conditions. These conditions are a starting velocity and acceleration equal to the ending velocity and acceleration of the support phase trajectory and an ending velocity and acceleration equal to the starting velocity and acceleration of the support phase. In short, for the ξ_x direction, this means

$$\xi_{x,f}(0) = \xi_{x,s}(T_s), \tag{5.13a}$$

$$\dot{\xi}_{x,f}(0) = \dot{\xi}_{x,s}(T_s), \tag{5.13b}$$

$$\ddot{\xi}_{x,f}(0) = \ddot{\xi}_{x,s}(T_s), \tag{5.13c}$$

$$\xi_{x,f}(T_f) = \xi_{x,s}(0), \tag{5.13d}$$

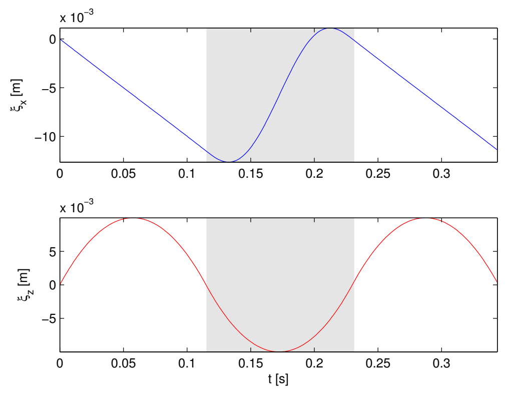
$$\dot{\xi}_{x,f}(T_f) = \dot{\xi}_{x,s}(0), \tag{5.13e}$$

$$\ddot{\xi}_{x,f}(T_f) = \ddot{\xi}_{x,s}(0), \tag{5.13f}$$

which results in the linear system

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & T_f & T_f^2 & T_f^3 & T_f^4 & T_f^5 \\ 0 & 1 & 2T_f & 3T_f^2 & 4T_f^3 & 5T_f^4 \\ 0 & 0 & 2 & 6T_f & 12T_f^2 & 20T_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} \xi_{x,s}(T_s) \\ \dot{\xi}_{x,s}(T_s) \\ \ddot{\xi}_{x,s}(T_s) \\ \xi_{x,s}(0) \\ \dot{\xi}_{x,s}(0) \\ \ddot{\xi}_{x,s}(0) \end{bmatrix} \tag{5.14}$$

which is solvable for $T_f \neq 0$, so in all practical cases. In the ξ_z direction polynomial, the coefficients are determined in the same way. Note that the time variable $\hat{t}$ goes from 0 to T_s in the support phase and from 0 to T_f in the flight phase. In other words, it is reset to zero after switching states, which simplifies expressions and decreases risk of bugs in programming. The quintic part of the reference signal is shown in the gray part of figure 5.2a and the negative part of the curve in figure 5.2b, completing the full trajectory.


(a) Time trajectory in ξ_x and ξ_z

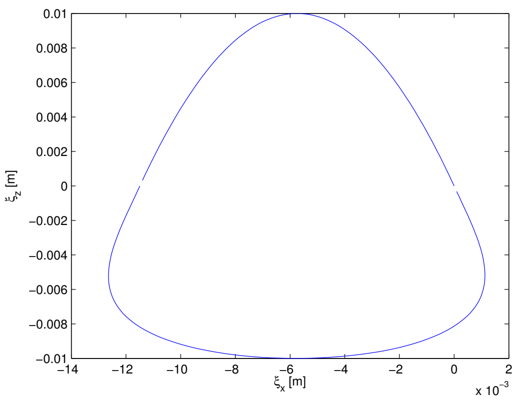

(b) Spatial trajectory: ξ_z vs ξ_x

Figure 5.2: *Jumping reference trajectory in the direction based reference frame for a frequency of* 4.0 Hz*, a flight time equal to the support time, an initial height of* 0*, a descent depth of* 1 cm *and a mean forward velocity of* 0.1 m/s*. The gray area in the time plot signifies the flight phase.*

5.3 Results

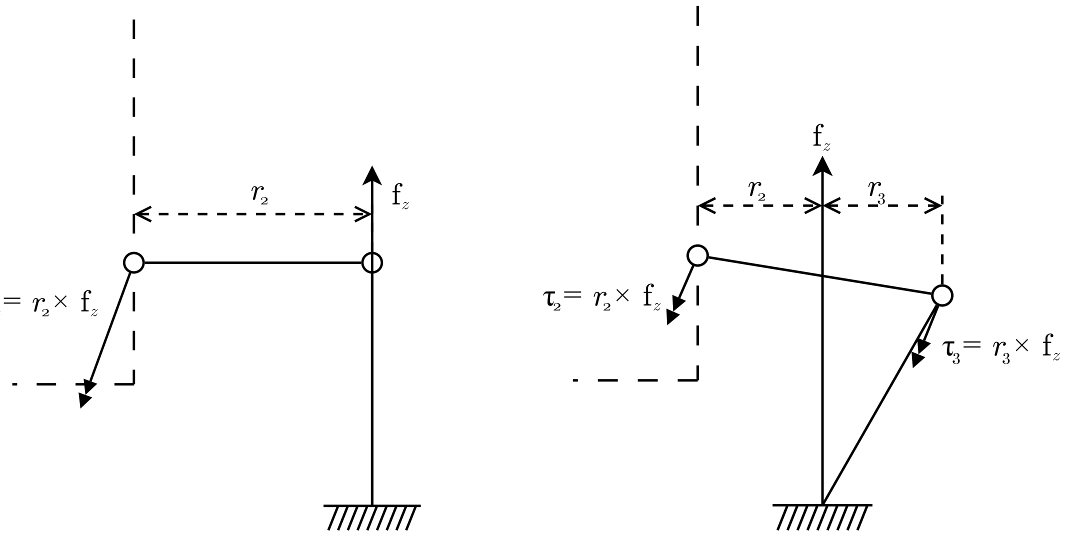
So far, switching between the flight and support phase is coordinated by a simple time delay. The support phase takes a calculated support time, but the flight phase is given a parameter as flight time. This is a very primitive feed forward implementation and only suitable for first tests. For more flexible behaviour, suitable for real-life situations, a measure for being in either the flight or the support phase should be implemented. This may be done using IMU data. As acceleration is always $\mathbf{g}^W = [0, 0, -9.81]^T$ during the flight phase, this may be accurate enough.

The assumption that the robot is a one, or two DoF system may be too simple in practice. The differences in leg length, spring constants or other real-world issues may have a non-neglectable influence on the body's orientation. A controller should then be designed to compensate for these differences and stablize the (switching) system.

Due to the time limitation to this project, no experiments were done while recording data. A simple test of the reference however, was done. This showed that the reference trajectory is followed and the frequency can be recognized when the robot feet are in the air. However, when the robot is placed on the ground, it cannot follow the trajectory any more. This is most likely caused by the lack of compensation for the robot's weight and the acceleration of the robot body. These forces are effectively a time-varying disturbance to every leg tip, resulting in bad tracking behaviour. A well-tuned feed forward component using the acceleration from the reference trajectory component could compensate for this disturbance, possibly giving better results.

During walking, but also during jumping tests, the cable between the coxa and the femur tends to break often. Breaking of cables may become an even bigger problem when jumping autonomously, as this causes the accelerations of the body, and thus the joint torques, to increase, so a solution must be found to reduce or resolve this problem. One possible solution may be to use stronger cables or another mechanism to transfer the torque from the torsional spring to the joint itself. Before redesigning the hardware, the leg configuration can best be chosen such that the joint torques are spread more evenly. A sketch of this is shown in figure 5.3. However, as said before, one should keep in mind that this could influence the resonance frequency.

In the future it may be interesting to try jumping on only three legs, which would imply a


r_2
f_z
$\tau_2 = r_2 \times f_z$
f_z
r_2
r_3
$\tau_2 = r_2 \times f_z$
$\tau_3 = r_3 \times f_z$

(a) Joint torque centered at second joint (b) Joint torques distributed over joints 2 and 3.

Figure 5.3: *Joint torque distributions with different leg configurations. The configuration in (a) is similar to the one used for the jumping so far. A better option may be configuration (b).*

resonance frequency of $\frac{1}{\sqrt{2}} f_{r,6}$. This can then be extended to running by alternating the set of legs in support phase. This results in a reference trajectory that has a typical frequency of $\frac{1}{2\sqrt{2}} f_{r,6}$ or lower, depending on the jump height, which is less demanding for the controllers.

Chapter 6

Conclusions and recommendations

The goal during this project was to revive the Hexapod, bringing it back to its old capabilities, and to explore further possibilities and limitations of the robot. In the beginning, the only documentation on the Hexapod was the report written by Willems [8], which is very abstract on the software part. Also the Wiki page was empty apart from outdated installation instructions. This made it difficult to get started without the introduction of a senior member of the Hexapod team. As no team existed at the start of this project, the lack of documentation was a problem. This added another goal to the project: documenting both the hardware and the software such that others, continuing work on the Hexapod, are able to get started quickly.

6.1 Results

To revive the robot, the software was updated such that it runs with a newer version of ROS: Fuerte. Problems with the hardware startup such as not having a large enough supply of current were solved and documented on the Wiki page. The recovered software was tested on the robot and also documented on the Wiki page. Fundamental improvements were made on the communication between Orocos components and the communication between Orocos and ROS was fixed. The safety component was rewritten such that the robot can now recover from a safety switch-off and all components were given an assertion to check whether they were given valid parameters or not. The theory behind the body dynamics was corrected as well as its implementation in the gravity compensation component. All of this made it possible to make the robot walk again.

Finally, also a start was made on making the Hexapod jump. The resonance frequency at a certain leg configuration was determined and a center of mass trajectory was generated. From this trajectory, a trajectory for the legs was generated to do a basic proof of concept. This proof of concept was not yet performed with succes, but with an additional feedforward on the forces needed to accelerate the body, the Hexapod may be able to jump. The results from this proof of concept should point out if it is necessary to control the jumping motion using an additional switching body controller.

6.2 Recommendations

Keeping in mind the goal of having the Hexapod play football on a real pitch, there are many things that remain to be done. In the future it would be practical if the Hexapod could replace an omni-wheeled TechUnited robot. For this to be realized, it is necessary to run parts of the software from these TURTLEs on the Hexapod. To do this, input from ROS should be received and interpreted by Orocos. The Orocos level software could then decide whether it is best to walk slowly, with a tripod gait, jump or run. Also the transition between these gaits should be handled.

Before testing jumping reference trajectories on the robot, it may be useful and safer to first test on the Gazebo robot simulator. To do this, the communication between Orocos and the simulator needs to be restored. Also improving safety when testing, the safety component may be improved with a dynamic limit to the coxa joints, ensuring that the legs do not hit each other.

To walk with a stable and regular gait, the trajectory generation, which is now separate for each leg, should either be centralized or coordinated by a central component. Also the implemented reference trajectory itself must be changed to make the robot walk more smoothly by reducing impact forces when getting in contact with the ground.

To jump and/or run, suitable joint feed forward needs to be designed. Also the ground contact perception should be improved for the robot to be able to distribute the forces to compensate for its own weight over its supporting legs. Finally, the hardware should be redesigned such that the robot is able to withstand the large joint torques arising from jumping and running.

Bibliography

[1] B Mrkajic. *Systems and Control Library Development.* PhD thesis, 2011. 12

[2] R.A. Ottervanger, M. di Giandomenico, and J.J.P.A. Willems. Hexapod Wiki page, 2014. 3

[3] Ian Postlethwaite and Sigurd Skogestad. *Multivariable Feedback Control.* Wiley & Sons, 2 edition, 2001. 8

[4] Peter Soetens. The Orocos Component Builder's Manual, 2002. 9

[5] Peter Soetens. The Deployment Component, 2006. 11

[6] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*, volume 141. Wiley & Sons, 1 edition, 2006. 7, 13, 20

[7] Barkan Ugurlu, Jody a. Saglia, Nikos G. Tsagarakis, and Darwin G. Caldwell. Hopping at the resonance frequency: A trajectory generation technique for bipedal robots with elastic joints. *IEEE International Conference on Robotics and Automation*, pages 1436–1443, May 2012. 19

[8] J.P.P.A. Willems. *Control of a hexapodal robot.* PhD thesis, Eindhoven University of Technology, Eindhoven, 2011. 3, 7, 8, 9, 10, 13, 15, 23

[9] R. Woering. *Simulating the "first steps" of a walking hexapod robot.* PhD thesis, Eindhoven University of Technology, 2010. 14, 15

Appendix A

Matlab

To process the data from the resonance frequency measurements, the Matlab code below was used.

```
function [f,X] = fftplota(x,t,Fs,a,b,name,c,fc)

% fftplota
%
%   SYNTAX
%   [f,X] = fftplota(x,t,Fs,a,b,name,c,fc)
%
%       X:    frequency spectrum vector
%       x:    signal vector
%       t:    time vector
%       Fs:   sample frequency
%       a:    toggle time plot
%       b:    toggle frequency plot
%       name: signal name
%       c:    plotting color string
%       fc:   cutoff frequency
%
%   DESCRIPTION
%   fftplota returns the frequency spectrum of and plots a displacement
%   signal vector x versus a time vector t and its frequency spectrum
%   using sample frequency Fs.
%   The name of the signal is used in the titels of the figures.
%   Axis labels may be altered to fit the needs of the user.
%   The function returns the amplitude vector and its corresponding
%   frequency vector. The plot toggles turn on or off displaying the time
%   and frequency plots.
%   The spectrum is plotted at least up to the user defined cuttof
%   frequency. When this frequency is chosen higher than the sampling
%   frequency, the base of the spectrum (the spectrum up to the sampling
%   frequency) is mirrored

T = 1/Fs;                              % Sample time
L = length(x);                         % Length of signal
if L ~= length(t)
    error('Input arguments x and t must be of same lengths')
else
    if a
        plot(t,x)
        title(['Time plot of ',name])
        xlabel('Time [s]')
        ylabel('Acceleration [m s^{-2}]')
        if b
            figure
        end
    end

    NFFT = 2^nextpow2(L);            % Next power of 2 from length of x
    n = ceil(fc/Fs);

    X = fft(x,NFFT)/L;
    f = Fs*linspace(0,n,n*NFFT);

    % stretch frequency spectrum by mirroring the spectrum in the sampling
    % frequency. Enough to plot the required frequency fc.
    Xm = fliplr(X);
    Xo = X;

    for i = 1:n
        if rem(i,2)
            X = [X Xm];
        else
            X = [X Xo];
        end
    end

    % Plot single-sided amplitude spectrum.
    if b
        semilogx(f(1:end/2),2*abs(X(1:(n*NFFT)/2)),c)
        title(['Single-Sided Amplitude Spectrum of ',name])
        xlabel('Frequency (Hz)')
        ylabel('Amplitude [m s^{-2}]')
    end
end
end
```