ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Bachelor-Thesis

Simulation and Animation of a Robotic Puppet

Spring Term 2013

Supervised by:
Dr. Paul Beardsley
Christian Gehring

Author:
Kevin Egger

Declaration of Originality

I hereby declare that the written work I have submitted entitled

Simulation and Animation of a Robotic Puppet

is original work which I alone have authored and which is written in my own words.[1]

Author(s)

Kevin	Egger

Supervising lecturer

Paul	Beardsley
Christian	Gehring

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`http://www.ethz.ch/students/exams/plagiarism_s_en.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Place and date	Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Contents

Abstract

This bachelor thesis was written within the focus project PuppetCopter. The PuppetCopter is a marionette suspended from a hexacopter. The puppet is actuated with a special system which is located below the hexacopter.

The goal of this thesis was to create a tool which allows the user to create performances for the PuppetCopter. The main focus here lied in the animation of the puppet. The desired movements were recorded in a motion capture environment. From this data, the commands for the motors of the puppet actuation system were calculated. These can then be started on the real system via an easy to use graphical user interface. To verify the motions without having the need to have the real system, a simulation of the puppet was created as well as part of this project. In this simulation, any effects of the copter were neglected.

Symbols

Symbols

a	scalar value
$\mathbf{a}$	vector
$\mathbf{A}$	matrix
A	point
α, β, γ	rotation angle around the x, y or z axis
i	vector index
k	discrete time index
$\mathbf{q}$	vector of generalized coordinates

Indices

act	actuation bar
arm	arm of the puppet
$body$	body of the puppet
$head$	head of the puppet
l	left side
leg	leg of the puppet
r	right side
s	string

Acronyms and Abbreviations

ASL	Autonomous Systems Lab
CAD	Computer Aided Design
COG	Center of Gravity
DOF	Degree of Freedom
ETH	Eidgenössische Technische Hochschule
ROS	Robot Operating System
ZHdK	Zurich University of the Arts

Chapter 1

Introduction

1.1 Motivation and Goals

The goal of this bachelor thesis is to create a tool to use for the focus project PuppetCopter which allows the user to create performances for the system. These performances primarily include movements patterns like walking or waving for the puppet. Additionally, a method to define waypoints for the copter is required to achieve a synchronized performance between the copter and the puppet. This method needs to be integrated in the same tool as the generation of the puppet motion in order to create a coordinated show.

The software developed in this thesis should therefore allow an easy definition of the desired movements. These motions need to get transformed to the corresponding motor commands which can then be sent to the actuation unit of the puppet. This allows the motors to move the puppet to the correct position and performing the desired movement.

To make testing of the calculated motor commands easier, a simulation of the puppet will be created as well. This allows the user to visualize the movements and provides a possibility to verify the computed motor commands without having the need to test it with the real system.

1.2 Focus Project PuppetCopter

In the third year of the bachelor programme in mechanical engineering at ETH Zurich, students get the choice to either enlarge their knowledge by visiting further lectures or can work in a small team on a project for a year. These teams consist of up to ten people and get to create a complete product from an initial concept to a first working prototype.

The team of the focus project PuppetCopter consists of eight people: Six students from ETH Zurich in their final year of the bachelor programme and two students in Industrial Design from ZHdK.

The PuppetCopter, which can be seen in Figure 1.1, consists of two main subsystems. On top, there is a hexacopter. Below is a marionette, the puppet, attached with strings. To move the puppet, there is an actuation platform located in between these two parts. The actuation system is able to move the strings of the marionette

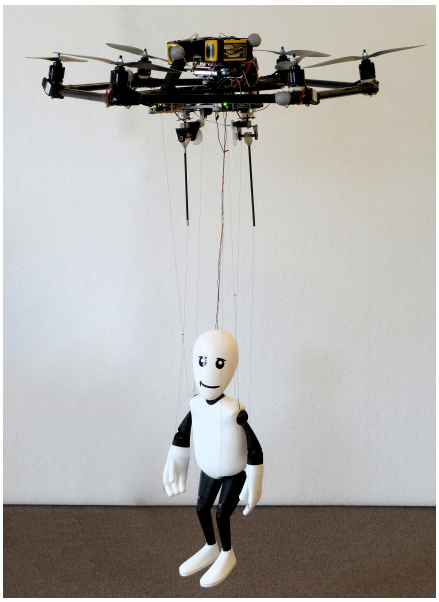


Figure 1.1: Overview of the PuppetCopter. On top, there is the copter. Located on the bottom is the puppet. In the middle, there is the actuation platform.

like a real puppeteer. Furthermore, a tilt compensation ensures that the actuation platform is always parallel to the ground and decouples the puppet from the copter. This simplifies the control of the marionette since the attitude of the copter does not need to be considered.

The reader is referred to [1] to get further details about the focus project PuppetCopter.

1.3 The Puppet

In this Section, the puppet is described. The puppet is the object that is going to be simulated in this thesis and can be seen in Figure 1.2.

The puppet is about 60 cm tall and weighs about 300 g. Its shape and limbs resemble these of a human. It is completely 3D-printed.

The puppet has fifteen DOFs, all of them are rotational. There are three DOFs in each shoulder and one in each elbow. Furthermore, one DOF is located in the hip as well as there is one in the knee. Two DOFs are located in the neck to allow head movements. The final fifteenth DOF is the rotation of the puppet's main body. This allows to move the puppet into a Superman-like flying position.

A more detailed look of the puppet with a focus on the mechanical model can be found in Section 2.3.

1.4 Actuation of the Puppet

Like with a traditional marionette, there are no motors inside the puppet itself. Everything is actuated from the actuation system located below the copter. The actuation system can be seen in Figure 1.3. Each arm, each leg as well as the head

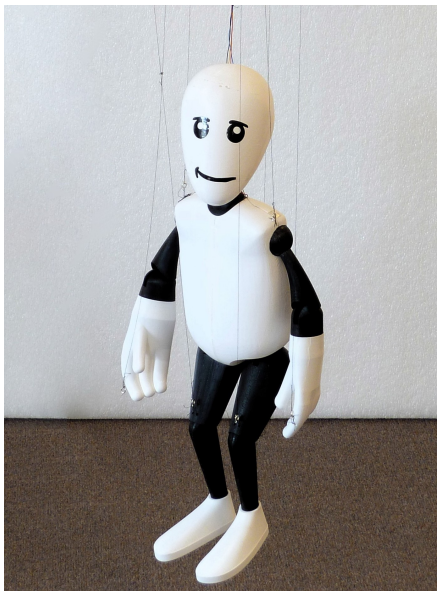


Figure 1.2: The puppet. It has fifteen DOFs: Four in each arm, two in each leg, two in the head and one to move up the body to go into a flying position.

is attached with a string to the actuation system, each limb attached to a separate actuation module. There is another string which goes to the back of both legs. This makes it possible to lift up the legs and the body of the puppet to go into a flying position. All these strings can be reeled up with a motor, changing the length of the string. There is one additional string on each shoulder. These two strings are fixed and cannot be moved. They are used to attach the puppet to the copter and carry the marionette's weight.

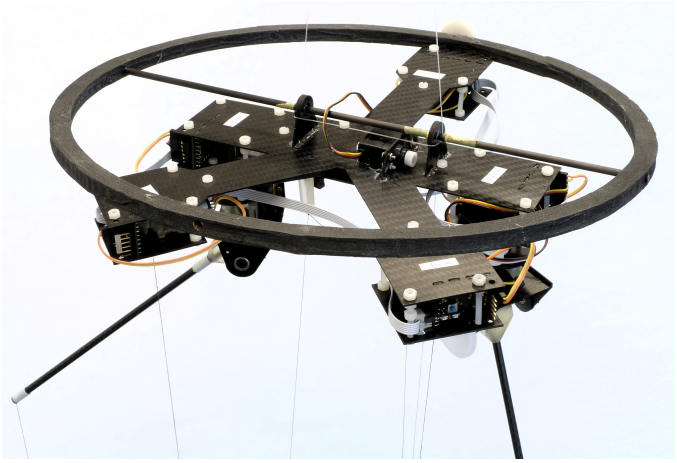


Figure 1.3: The complete actuation system for the puppet which is located directly below the copter.

Taking inspiration from real puppeteers who can freely move the string in 3D space, one actuation module is extended with two servo motors. Additionally, the string is first guided through a small, moveable bar. The first servo motor allows for a rotation around the z axis, while the second one is used to move the bar up and down. This then results in a 3D movement of the end point of the actuation bar. One such actuation module can be seen in Figure 1.4.

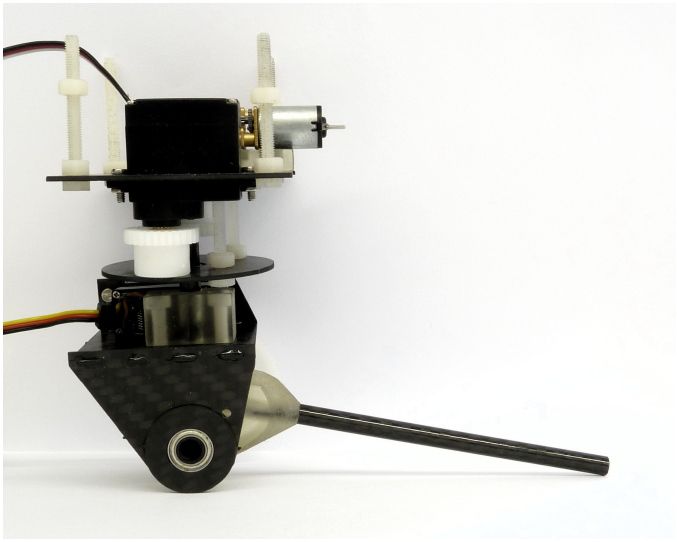


Figure 1.4: A three dimensional actuation module. It consists of two rotational DOFs to move the actuation bar around. The third DOF is realized by reeling up the string.

Since not all limbs in the puppet need to be moved with three DOFs, some of the actuation modules are reduced to only one DOF which is changing the string length. For the legs, only such a reduced module is needed. The arms and the head are actuated with three DOFs.

More on the mechanical and electrical design of the actuation system can be found in [5].

Chapter 2

Simulation

In this chapter, the simulation is discussed. First, the requirements and the assumptions made are introduced. Next, various tools for creating the simulation have been analysed. Afterwards, the mechanical model of the puppet is described in detail. As a last point, the implementation of the simulation is explained.

2.1 Requirements and Assumptions

The first question which has to be asked is whether the simulation must be dynamic. In the case of this project, a dynamic simulation is preferred. The reason for this is that some limbs of the puppet, like the lower part of the leg, are not actuated and would not move at all in a non-dynamic simulation.

Furthermore, the strings have to be modelled in the simulation somehow. It also needs to be capable to read in an input in order to be able to simulate the behaviour under the inputs the actuation system gets.

Additionally, several assumptions were made to simplify the problem:

- The movements of the copter are neglected. This means that the actuation platform is fixed in space for the simulation.
- The puppet is rigidly connected to the copter, i.e. the puppet's shoulder always stays at the same position.
- All strings are modelled as forces acting on the limbs. This means that the strings can push the limb, unlike their real-world counterparts.

2.2 Tool Evaluation

For creating a simulation, various tools have been considered. The tool needs to be capable of building a model of the puppet with the present DOFs as well as some easy ways to correctly replicate the influences of reeling up a string. The simulation tool must also provide for a method to visualize the calculated results. The considered tools are listed with their respective advantages and disadvantages below.

2.2.1 Gazebo

Gazebo[1] is a program for simulating robots. The model is specified in an XML file. The software also features a visualization of the model's movements. To interact with the simulation, it is possible to write `C++` plugins. Furthermore, Gazebo can be easily connected to a ROS[2] environment which is already used for the communication on the copter. This would make it easy to exchange the simulation with the real system.

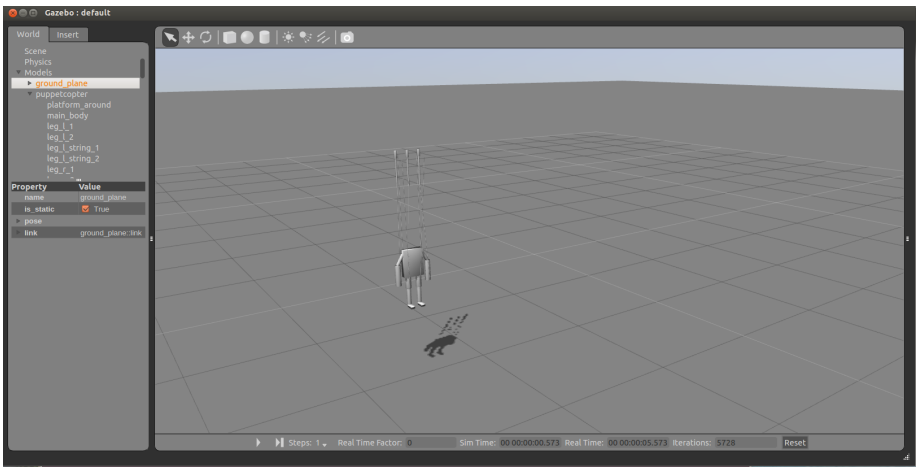


Figure 2.1: Gazebo Simulation Environment.

2.2.2 MATLAB and/or C++

This option would not rely on any already implemented software. This gives complete control over the simulation which makes it adjustable to the project's needs. However, this also means that every component would have to be reimplemented: The equations of motion have to be defined. Then these equations need to get solved numerically. Finally, a visualization of the simulated movements has to be created.

2.2.3 proNEu and C++

Some of the parts of creating the simulation could be done by already existing software. The MATLAB tool proNEu[3], which was developed at the ASL, takes the kinematic tree of the model as an input and then generates the equations of motions.

These equations would then need to be implemented in a `C++` program. Furthermore, a visualization of the results also needs to be created.

This method also gives a lot of flexibility since the simulation can be adjusted as needed. The hardest part, the generation of the equations of motions, would be done by an already existing, simple to use MATLAB program.

[1] Website: http://gazebosim.org/ (May 2013)
[2] Website: http://www.ros.org/wiki/ (June 2013)
[3] Website: http://www.leggedrobotics.ethz.ch/doku.php?id=research:software (June 2013)

2.2.4 Motion Simulation with Siemens NX

Siemens NX is the CAD software which was used in the project. It also includes a motion simulation package. It allows to directly import a CAD model and simulate it with a visualization using the real model. However, as it is a closed system, an integration with an outside system is difficult to achieve.

2.2.5 SimMechanics

SimMechanics[4] is a MATLAB/Simulink tool. It allows to link multiple rigid bodies in the Simulink environment and can afterwards simulate it. However, it is difficult to model the strings and link the Simulink part to input commands from the outside.

2.2.6 Conclusion

Table 2.1 provides an overview of the considered tools. In the end, two of these solutions were taken under further consideration. The choice was between either Gazebo or proNEu. The other options were not considered any further because they either would need too much effort to implement or they do not provide a simple way to interact with the simulation from the outside.

	Gazebo	**MATLAB / C++**	**proNEu / C++**	**Siemens NX**	**SimMechanics**
Simulation Available	yes	no	partially	yes	yes
Visualization	yes	no	no	yes	yes
Integration in Custom Software	no	yes	yes	no	partially
Extensibility	yes	yes	yes	no	partially

Table 2.1: Overview of the considered tools to perform the simulation.

Gazebo would have been a good solution since everything is already programmed and there exists an easy way of interacting with the simulation. However, the simulation often did not respond to a force set on the limb. Such problems are hard to solve since it is uncertain if the problem lies in the software itself or in the given model.

For this reasons, it was decided to create the simulation with proNEu. This requires an additional program to solve the equations provided by proNEu and visualize these results. Fortunately, these steps are not difficult to implement. Furthermore, this allows to customize the simulation to exactly fit the needs of the project.

2.3 Mechanical Model of the Puppet

The following Section discusses the mechanical model of the puppet. First, it is explained how the mechanical properties of the puppet were found. Second, the

[4]Website: http://www.mathworks.ch/products/simmechanics/ (March 2013)

model of the puppet used to implement the simulation is presented.

2.3.1 Mechanical Properties

In order to be able to create the simulation as close to reality as possible, the mechanical properties of the puppet like mass, inertias, COGs as well as lengths and dimensions, have to be found.

Measuring the lengths of the different limbs can easily be done using the real puppet. The inertias, however, cannot be measured this simply. They have to be found using a CAD software. For this, the CAD models of the actual puppet were used.

In order to use the CAD software to calculate the mass and inertia properties as well as the location of the COGs, the density of the material has to be found first. A part of the puppet, one arm[5], was measured to have a weight of 13.3 g. With the CAD software, the volume of the same part was found as 13 437.8 mm^3. The density of the puppet's material can then be calculated by

$$\rho_{puppet} = \frac{m_{puppet}}{V_{puppet}} = 982 \, \frac{\mathrm{kg}}{\mathrm{m}^3}. \tag{2.1}$$

With this result, the mass and inertia values and the COG locations can be calculated. These results can be found in Appendix A.

2.3.2 Notation and Coordinate Frames

The following Section provide sketches with the mechanical model of the puppet. The following colour scheme is used

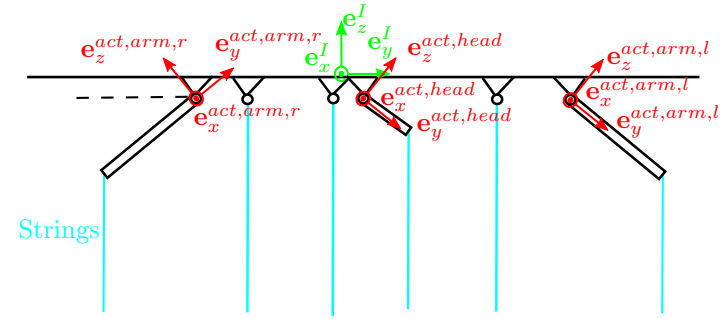
Green	Coordinate Frame
Orange	Coordinate Frame
Red	Coordinate Frame
Dark Blue	Degree of Freedom
Purple	Point
Light Blue	String

Actuation Platform

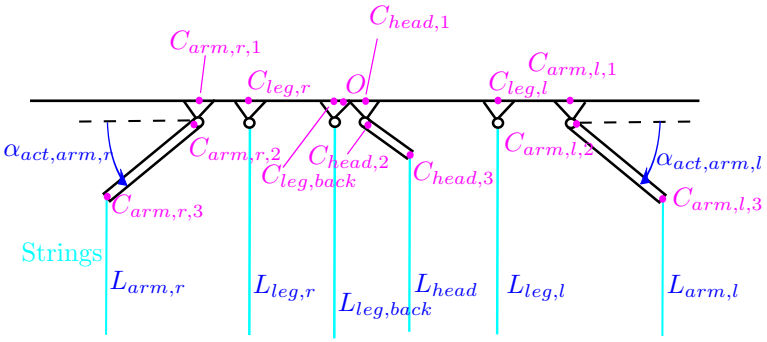
The mechanical model of the actuation platform is given in Figure 2.2. Figure 2.2(c) shows the coordinate frames of the actuation unit. In Figure 2.2(d), the DOFs of the actuation platform as well as all points are indicated.

As mentioned in Section 2.1, the actuation platform is assumed to be fixed in space. Therefore, the coordinate origin O as well as the inertial frame I is located there. There are three one dimensional actuation modules used to actuate the legs. They can only change the length of the strings. Furthermore, there are three actuation modules with three DOFs. These are used to control the head and the arms. They are able to change the string length and have to further rotational DOFs as indicated in Figure 2.2.

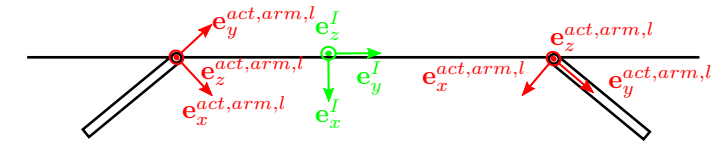
[5]The measured part consists of the black parts of the arm. It included both rigid body which form the arm, but without the hand piece.



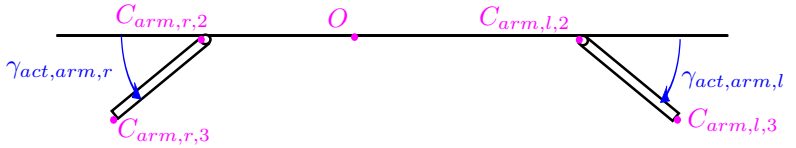
(a) Coordinates Frame from the front



(b) Point and DOFs from the front



(c) Coordinates Frame from the top (only with actuation units of arm)



(d) Point and DOFs from the top

Figure 2.2: Mechanical model of the actuation platform.

Body and Head

Figure 2.3 shows the mechanical model of the body and the head. On both sides of the shoulder, the immovable string going to the actuation platform is attached. A movable string is attached on the head.

This part of the puppet consists of three DOFs due to its joints. The first one is achieved by rotating the whole body around the y axis. There are two additional DOFs between the body and the head as indicated in the drawing. The direction of the angle is defined as drawn in Figure 2.3, the zero position is at the shown position.

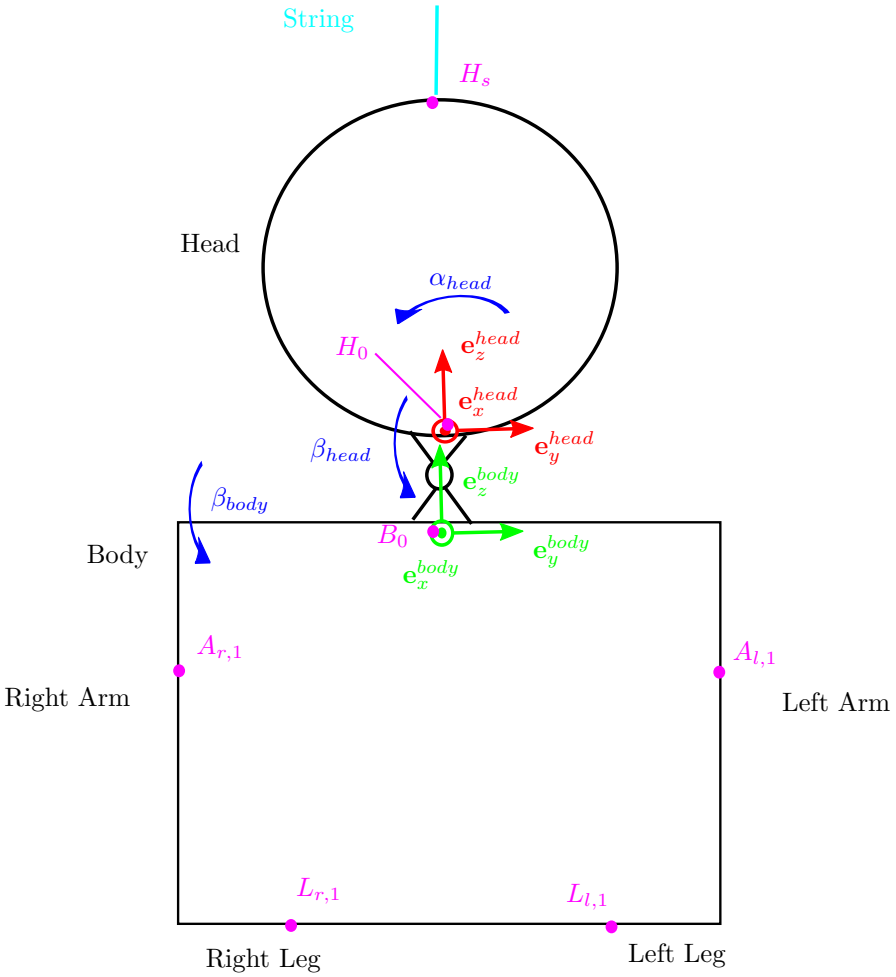


Figure 2.3: Mechanical model of the puppet's body and the head. The body is drawn shorter than in reality in order to reduce the size of the Figure.

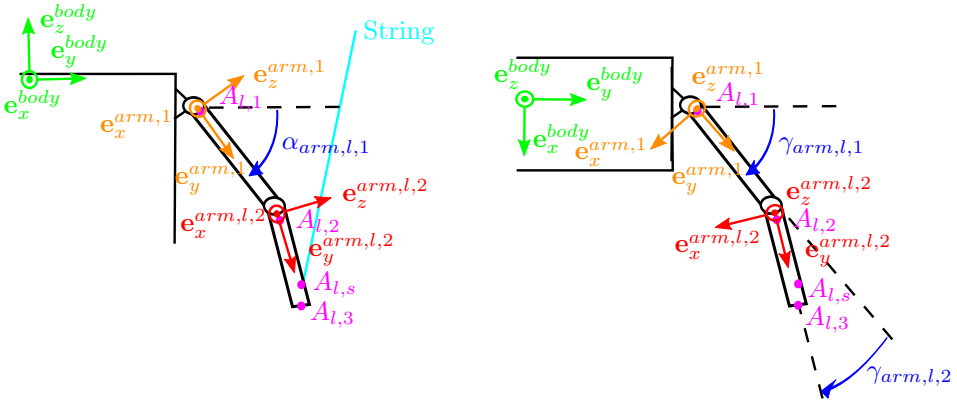


Figure 2.4: Mechanical model of the left arm.

Arms

In Figure 2.4, the mechanical model of the left arm can be seen. The same model applies to the right arm in a similar way. The coordinate frames on the right arm

are flipped however. This means that the direction of the y axis is reversed.

There is one string attached on the arm in the point $A_{l,s}$. It is located near the end of the arm. The arm consists of two rigid bodies. The upper body is connected to the main body of the puppet with a ball joint. This joint has three DOFs. Two of them are indicated in Figure 2.4. The third DOF is the rotation around the axis of the arm (y axis) with the angle $\beta_{arm,l,1}$. The connection between the lower and the upper part of the arm consists of only one DOF.

Legs

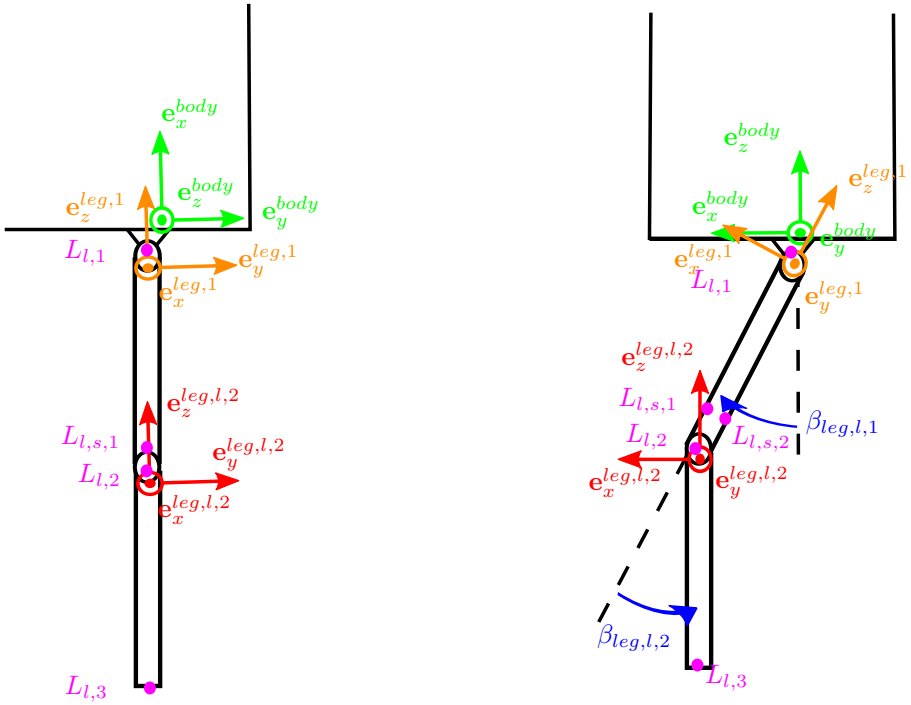


Figure 2.5: Mechanical model of the left leg.

The mechanical model of the left leg can be seen in Figure 2.5. The same model applies analogously for the right leg.

There are two strings attached per leg. Both of them are located directly above the knee joint. The first string is in front of the puppet. It is used for all leg movements like walking. The second string is attached on the back of the limb. It is used to lift the puppet up to go in a flying position, affecting the DOF β_{body} of the body (see 2.3). The back strings of both legs are controlled simultaneously.

A leg consists of two DOFs. There is one rotational DOF in the hip and another one in the knee.

2.4 Implementation

As a first step, a MATLAB program was written to generate the equations of motion with the help of proNEu. The kinematic tree of the puppet was built according to the model introduced in the previous Section. The output of proNEu are the matrix

$\mathbf{M}$ as well as the vectors $\mathbf{b}$ and $\mathbf{g}$ of the equations of motion[6]

$$\mathbf{M}(\mathbf{q}) \cdot \ddot{\mathbf{q}} + \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \mathbf{S}^\top \tau \tag{2.2}$$

where (see also [4])

$\mathbf{q}$	vector of generalized coordinates
$\mathbf{M}$	mass matrix $\in \mathbb{R}^{15\times 15}$
$\mathbf{b}$	coriolis and centrifugal components $\in \mathbb{R}^{15\times 1}$
$\mathbf{g}$	gravitational components $\in \mathbb{R}^{15\times 1}$
$\mathbf{S}$	selection matrix of the actuated joint $\in \mathbb{R}^{36\times 15}$
τ	generalized forces $\in \mathbb{R}^{36\times 1}$

The vector of the generalized coordinates is defined as

$$\begin{aligned} \mathbf{q} \;=\; \big(\; & \beta_{body} \quad \beta_{leg,l,1} \quad \beta_{leg,l,2} \quad \beta_{leg,r,1} \quad \beta_{leg,r,2} \quad \dots \\ & \alpha_{arm,l,1} \quad \beta_{arm,l,1} \quad \gamma_{arm,l,1} \quad \gamma_{arm,l,2} \quad \dots \\ & \alpha_{arm,r,1} \quad \beta_{arm,r,1} \quad \gamma_{arm,r,1} \quad \gamma_{arm,r,2} \quad \dots \\ & \alpha_{head} \quad \beta_{head} \;\big)^\top . \end{aligned} \tag{2.3}$$

The vector of the generalized forces is given by

$$\begin{aligned} \tau \;=\; \big(\; & \tau_{\beta_{body}} \quad \tau_{\beta_{leg,l,1}} \quad \tau_{\beta_{leg,l,2}} \quad \tau_{\beta_{leg,r,1}} \quad \tau_{beta_{leg,r,2}} \quad \dots \\ & \tau_{\alpha_{arm,l,1}} \quad \tau_{\beta_{arm,l,1}} \quad \tau_{\gamma_{arm,l,1}} \quad \tau_{\gamma_{arm,l,2}} \dots \\ & \tau_{\alpha_{arm,r,1}} \quad \tau_{\beta_{arm,r,1}} \quad \tau_{\gamma_{arm,r,1}} \quad \tau_{\gamma_{arm,r,2}} \dots \\ & \tau_{\alpha_{head}} \quad \tau_{\beta_{head}} \quad \dots \\ & F_{leg,l,1,x} \quad F_{leg,l,1,y} \quad F_{leg,l,1,z} \quad F_{leg,l,2,x} \quad F_{leg,l,2,y} \quad F_{leg,l,2,z} \quad \dots \\ & F_{leg,r,1,x} \quad F_{leg,r,1,y} \quad F_{leg,r,1,z} \quad F_{leg,r,2,x} \quad F_{leg,r,2,y} \quad F_{leg,r,2,z} \quad \dots \\ & F_{arm,l,x} \quad F_{arm,l,y} \quad F_{arm,l,z} \quad F_{arm,r,x} \quad F_{arm,r,y} \quad F_{arm,r,z} \quad \dots \\ & F_{head,x} \quad F_{head,y} \quad F_{head,z} \;\big)^\top \end{aligned} \tag{2.4}$$

where the τ_i are the joint frictions and the F_j the forces caused by the strings.

The selection matrix $\mathbf{S}$ is chosen as

$$\mathbf{S} = \begin{pmatrix} \mathbb{I}_{15} \\ \mathbf{J}_{leg,l,1}(\mathbf{q}) \\ \mathbf{J}_{leg,l,2}(\mathbf{q}) \\ \mathbf{J}_{leg,r,1}(\mathbf{q}) \\ \mathbf{J}_{leg,r,2}(\mathbf{q}) \\ \mathbf{J}_{arm,l,1}(\mathbf{q}) \\ \mathbf{J}_{arm,r,1}(\mathbf{q}) \\ \mathbf{J}_{head,1}(\mathbf{q}) \end{pmatrix} \tag{2.5}$$

where $\mathbb{I}_{15} \in \mathbb{R}^{15\times 15}$ is the identity matrix and

$$\mathbf{J}_i = \frac{\partial \mathbf{r}(\mathbf{q})}{\partial \mathbf{q}} \quad , \quad \mathbf{J}_i \in \mathbb{R}^{3\times 15} \tag{2.6}$$

are the Jacobians for the points where the strings are attached on the puppet.

[6]The tool proNEu also is also capable of generating the matrix $\mathbf{S}$. However, it was decided to create this matrix using Jacobians as discussed later and not use the result of proNEu.

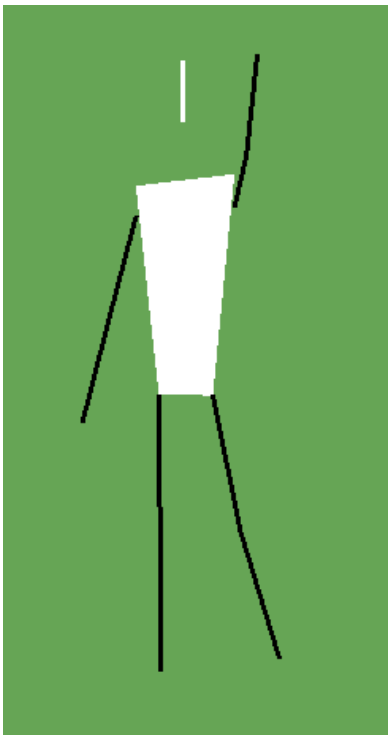


Figure 2.6: The implemented simulation visualized with the help of OpenGL.

The equation (2.2) is then implemented in a `C++` program and solved. To work with matrices and vectors in the `C++` code, the matrix library Eigen[7] is used. The equations of motion can then be solved after $\ddot{\mathbf{q}}$ with the aid of the before-mentioned library.

This solution is then integrated by using two Euler Forward integrators:

$$\ddot{\mathbf{q}}_{k+1} = \mathbf{M}^{-1}(\mathbf{q}_k) \cdot \left(\mathbf{S}^{\top}(\mathbf{q}_k)\tau(\mathbf{q}_k) - \mathbf{b}(\mathbf{q}_k, \dot{\mathbf{q}}_i) - \mathbf{g}(\mathbf{q}_i)\right) \tag{2.7}$$

$$\dot{\mathbf{q}}_{k+1} = \dot{\mathbf{q}}_k + \Delta t \cdot \ddot{\mathbf{q}}_{k+1} \tag{2.8}$$

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \Delta t \cdot \dot{\mathbf{q}}_{k+1} \tag{2.9}$$

where Δt is the time step which needs to be small in order for the result to be accurate. For the simulation, this time step was chosen as $\Delta t = 10^{-5}$ s.

The head is currently not fully implemented. Within the project PuppetCopter, it was decided that the main focus is a correct implementation of the arms and legs as well as the flying movement. While head movements are included in the equations of motion, there is no string or other external force acting on the head. This goes together with the actuation platform where there is also no actuation for the head yet.

To visualize the simulation, OpenGL[8] was used as can be seen in Figure 2.6. This visualization was then incorporated into a graphical user interface. The reader is referred to Section 3.3 for more details about the user interface.

While the simulation is performing, it gets the current input of the actuation system. As mentioned above, the strings are simplified in the simulation. The strings are

[7] Website: http://eigen.tuxfamily.org/index.php?title=Main_Page (May 2013)
[8] OpenGL is an 2D and 3D graphics library. Website: http://www.opengl.org/ (June 2013)

modelled as forces acting on the limbs, meaning that they can not only pull, but also push the body. To control these string forces in the simulation, a PID controller was programmed which gets the calculated string length as the reference position.

For the rotational DOFs of the actuation system, the angles get just set. To introduce the effects of these movements on the puppet, the following force was added:

$$\mathbf{F}_{act,angle}(k) = f \cdot \left(\mathbf{r}_{C_{act,3}}(k) - \mathbf{r}_{C_{act,3}}(k-1)\right) \tag{2.10}$$

with f being a constant factor and $\mathbf{r}_{C_{act,3}}$ being the position of the end of the actuation bar.

To prevent the puppet's belly from swinging too much, movements of it are disabled. They can be turned on by choice since they are required in order to move the belly up to simulate the flying movement.

Furthermore, there is damping in the joint. This damping was realized by

$$\tau_{i,damping} = -\delta_{joint} \cdot \dot{\mathbf{q}}_i \tag{2.11}$$

where δ_{joint} is the damping constant.

To prevent joints from exceeding their physical limits, another force was introduced. This force consists of a force and a damper:

$$\tau_{i,limit} = \begin{cases} -\left(c_{limit} \cdot (\mathbf{q}_i - \varphi_{i,upper}) + \delta_{limit} \cdot \dot{\mathbf{q}}_i\right) & \text{if } \mathbf{q}_i > \varphi_{i,upper} \\ -\left(c_{limit} \cdot (\mathbf{q}_i - \varphi_{i,lower}) + \delta_{limit} \cdot \dot{\mathbf{q}}_i\right) & \text{if } \mathbf{q}_i < \varphi_{i,lower} \end{cases} \tag{2.12}$$

where $\varphi_{i,lower}$ and $\varphi_{i,upper}$ are the joint limits (see Table 2.2), c_{limit} is the spring constant and δ_{limit} is the damping constant.

i	**Joint Name**	**Lower Limit [rad]**	**Upper Limit [rad]**
0	β_{body}	-1.6	1.6
1	$\beta_{leg,l,1}$	-0.25	1.5708
2	$\beta_{leg,l,2}$	-1.5708	0.1
3	$\beta_{leg,r,1}$	-0.25	1.5708
4	$\beta_{leg,r,2}$	-1.5708	0.1
5	$\alpha_{arm,l,1}$	-1.5	1.0472
6	$\beta_{arm,l,1}$	-3.14	3.14
7	$\gamma_{arm,l,1}$	-0.1	1.856
8	$\gamma_{arm,l,2}$	-1.5708	0
9	$\alpha_{arm,r,1}$	-1.5	1.0472
10	$\beta_{arm,r,1}$	-3.14	3.14
11	$\gamma_{arm,r,1}$	-0.1	1.856
12	$\gamma_{arm,r,2}$	0	1.5708
13	α_{head}	not implemented yet	
14	β_{head}	not implemented yet	

Table 2.2: The joint limits in the simulation. These were measured on the real system.

After the simulation has finished, the results get stored in an external text file, allowing for a later analysis of the simulation in MATLAB or a different tool. The content of this file is listed in Appendix B.2.

Chapter 3

Motion Planning

In this Chapter, it is shown how the movements are created. Figure 3.1 gives an overview of the process. First, the desired motions get defined (Section 3.1). From this input, the needed motor commands for the actuation are calculated (Section 3.2). This data can then be either verified in a simulation (see Chapter 2) or sent to the real system (see [5] for the puppet and [3] for the copter). The calculation of the actuation commands and the simulation are integrated in the same software. These two elements can be accessed from the graphical user interface (Section 3.3).

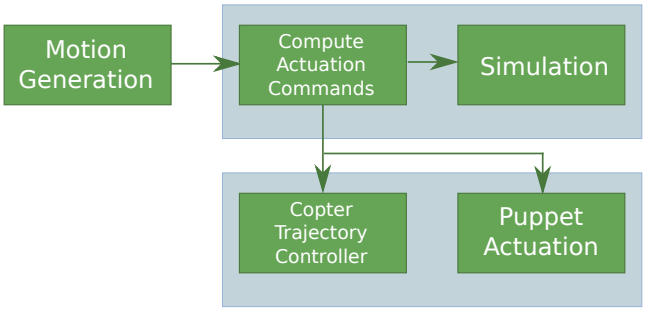


Figure 3.1: Overview of the process of creating a motion. First, the desired movements get defined. Then, the corresponding actuation commands are calculated. Finally, these can be either verified in a simulation or sent to the real system.

3.1 Movement Generation

In order to run the PuppetCopter, the desired motions have to be defined. There exist several possible approaches for doing this. One option would be to construct a second puppet with the same DOFs as the original one, but with encoders built in its joints. The puppet could be then placed in a desired position while the joint angles are read out.

Another option would be to use human motion data. This data would have to be transformed to fit the puppet. Furthermore, it would be necessary to determine if this data consists movement that are feasible for the puppet.

Another possibility would be to use a motion capture system. Here, the OptiTrack system could be used. The motion capture works the following way: There are

several cameras attached in the room. These cameras track the position of reflective marker balls which have to be put onto the object. The big advantage of this setup is that it is possible to record the movements with the actual puppet. This ensures a movement that is physically possible with the real marionette. Furthermore, the movements could be played by an actual puppeteer. Another advantage of this option is that a collision detection is not needed since the movements are recorded with the actual system. Collisions can already be avoided during the recording.

The disadvantages of this approach are, since the system is camera-based, that special care needs to be taken that all the markers on the puppet are visible for all the time. It is easily possible that the system loses one of the markers. This is usually solved by attaching multiple markers on the same body. However, this is difficult to implement with such a small object.

It was decided to work with the motion capture system since with this setup, the procedure of recording the motions is similar to playing an actual marionette. To do this, each actuated part of the marionette – each leg, each arm and the head – gets a marker ball attached at the same place where the string is attached. On the main body of the puppet however, three markers are used. This allows an additional tracking of the orientation of the marionette. The puppet equipped with the markers can be seen in Figure 3.2.

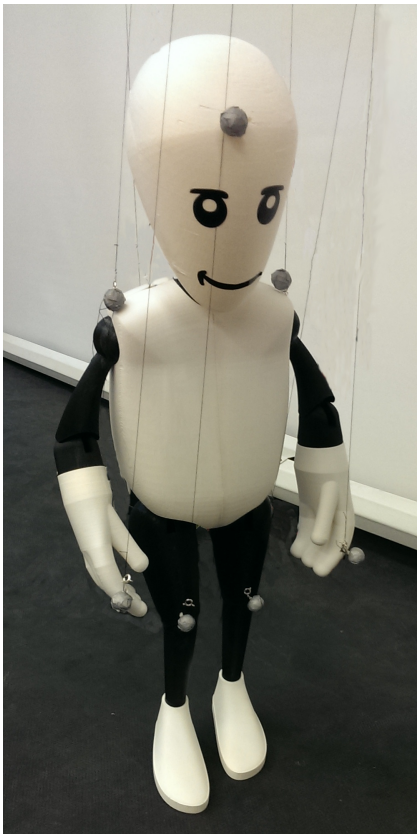


Figure 3.2: The puppet equipped with reflective markers for the use with the motion capture system.

To play the puppet, it will not be attached to the actuation platform used to autonomously control the marionette, but instead to a device that is used by puppeteers, called control bar. A control bar can be seen in Figure 3.3. Another set of

Figure 3.3: The marionette control bar used for the motion capture.

three markers is positioned on the control bar to simultaneously record the desired trajectory of the copter.

After recording the movements, the tracked marker positions can be exported as a `.c3d` file. With a free tool from the Carnegie Mellon University[1], this data can be loaded into MATLAB. There, the captured data can be saved into a file that can be read in the `C++` program.

The MATLAB tool loads the data into a three dimensional array – frame (time), marker, and coordinate (x,y,z). This data can be converted to a two dimensional structure which can be easily written to a file.

Since the coordinate frame of the motion capture system does not correspond with the puppet's coordinate frame, the position data has to be modified. While the z axis from both frames show in the same direction, only one rotation is necessary to transform into the other frame as can be seen in Figure 3.4.

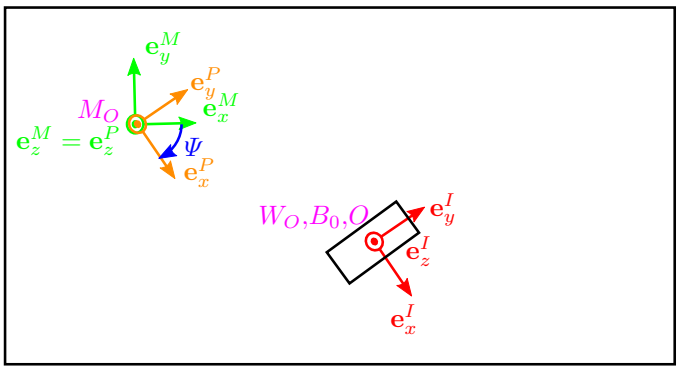


Figure 3.4: Different coordinate frames from the motion capture. M system (green): Motion capture coordinate frame; P system (orange): Puppet coordinate frame rotated to have axes correctly aligned; I system (red): Frame on actuation platform. Points: M_O: Motion capture origin (on ground); O: origin of I frame (on actuation platform); B_0: origin of puppet body coordinate frame (on puppet body); W_O: point below O on ground.

The rotation angle Ψ between the two coordinate frames is defined as

$$\Psi = \arctan\left(\frac{\Delta y}{\Delta x}\right) + \frac{\pi}{2} \tag{3.1}$$

[1] Website: http://mocap.cs.cmu.edu/ (May 2013)

where $\Delta x = x_r - x_l$ and $\Delta y = y_r - y_l$ with (x_r, y_r) the position of the marker on the right shoulder and (x_l, y_l) the position of the marker on the left shoulder.

The other tracked points can then be easily rotated around the z axis with the angle Ψ by

$$ {}_P\mathbf{r} = A_{PM}(\Psi) \cdot {}_M\mathbf{r} \tag{3.2}$$

where $A_{PM}(\Psi)$ is the rotation matrix defined as

$$ A_{PM} = \begin{pmatrix} \cos\Psi & \sin\Psi & 0 \\ -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.3}$$

Any of the tracked points on the puppet (indicated with P_s) is then given by

$$ \mathbf{r}_{W_O P_s} = \underbrace{\mathbf{r}_{M_O P_s}}_{\text{From motion capture}} - \underbrace{\mathbf{r}_{M_O B_0}}_{\text{From motion capture}} + \mathbf{r}_{W_O B_0} \tag{3.4}$$

where $\mathbf{r}_{W_O B_0} = \begin{pmatrix} 0 & 0 & z_{body} \end{pmatrix}^\top$. In all of the following calculations, the P coordinate frame is used, with the coordinate origin moved to W_O.

The yaw angle of the reference copter, the marionette control bar, can be calculated in a similar manner:

$$ \Psi_{Copter} = \arctan\left(\frac{x_r - x_l}{y_r - y_l}\right) - \Psi_0 \tag{3.5}$$

where (x_r, y_r) is the position of the marker on the right side of the control bar and (x_l, y_l) the position of the left side.

The values for the copter trajectory is saved relative to its starting point $(x_0, y_0) = (x(t=0), y(t=0))$ as well as $\Psi_0 = \Psi(t=0)$. The z position of the copter will be kept at the same level as it is recorded in the motion capture system.

With all the calculations done, the output file can be written. This file is a list of all values separated by a tab stop (`'\t'`) and each line representing another frame. The saved file has the extension `.pcr` and contains all the necessary information for the performance. The contents of each column can be seen in Appendix B.1.

3.2 Actuation Commands

With the desired puppet motion defined, the commands for the actuation system can be found. In this section, it is shown how the recorded puppet position can be transformed into the actuation position, i.e. the string length and two angles. As mentioned in Section 2.4, the head is not yet actuated by the system. However, the calculation for the actuation commands is analogously to that of the arm since both actuation units have the same DOFs.

3.2.1 Legs

Since the legs are only actuated one dimensionally, the calculation of the corresponding motor positions is very simple. Here, only the transformation for the left leg will be shown. However, it is analogously for the right leg.

The actuation unit for the left leg is positioned at

$$\mathbf{r}_{W_O C_{leg,l}} = \begin{pmatrix} t_{x,act,leg} \\ t_{y,act,leg,l} \\ z_{body} + L_{shoulder} - h_{act} \end{pmatrix}. \tag{3.6}$$

The recorded data has the left leg positioned at

$$\mathbf{r}_{W_O L_{ls}} = \begin{pmatrix} x_{leg,l} \\ y_{leg,l} \\ z_{leg,l} \end{pmatrix}. \tag{3.7}$$

With this, the length of the string can be calculated as

$$L_{leg,l} = \left\| \mathbf{r}_{W_O C_{leg,l}} - \mathbf{r}_{W_O L_{ls}} \right\|. \tag{3.8}$$

3.2.2 Arms

The calculation of the motor positions for the arms is more difficult than for the legs since the arms are attached to a three dimensional actuation module. The following approach is used:

1. Read in the input which is the desired position of the arm, or more specifically, the desired position of the point where the string is attached (initial position, see Figure 3.5(a)).
2. Determine the first rotational angle, $\gamma_{act,arm,l}$, by rotating the actuation bar around the z axis until it is above the puppet's arm (see Figure 3.5(b)).
3. Next, the second angle, $\alpha_{act,arm}$, will be calculated by rotating the actuation bar downwards until the end of the actuation bar is directly above the puppet's arm, creating a straight line between these two points (see Figure 3.5(c)).
4. The length of the string can then be found as the distance between these two points.

Step 1 The tracked point of the arm is given by

$$\mathbf{r}_{W_O A_{l,s}} = \begin{pmatrix} x_{arm,l} \\ y_{arm,l} \\ z_{arm,l} \end{pmatrix} \tag{3.9}$$

The base point of the actuation unit is located at

$$\mathbf{r}_{W_O C_{arm,l,2}} = \begin{pmatrix} t_{x,act,arm,l} \\ t_{y,act,arm,l} \\ z_{body} + L_{shoulder} - h_{act,arm} \end{pmatrix} \tag{3.10}$$

Step 2 The first angle can be found as

$$\gamma_{act,arm,l} = \arctan\left(\frac{x_{arm,l} - t_{x,act,arm,l}}{y_{arm,l} - t_{y,act,arm,l}} \right) \tag{3.11}$$

To prevent a collision of the actuation bar and the strings of the leg, this angle will be limited to $\gamma_{act,arm,l} \in [0, 1.4]$.

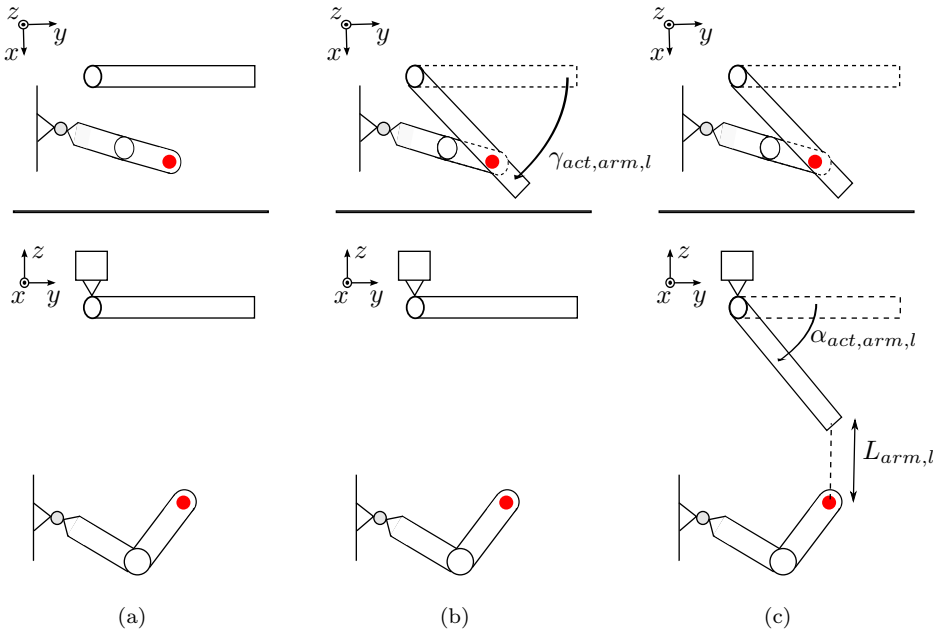


Figure 3.5: Calculation of the actuation commands for the left arm. The top row gives a view from top, the bottom row shows the situation from the front. The red dot indicates the tracked point on the arm where the string is attached. On the top is the actuation unit, on the bottom lies the puppet's arm. (a) Initial position. (b) The actuation bar is rotated around the z axis until it is above the puppet arm. (c) The actuation bar is rotated downward, creating a straight connection for the string between the limb and the end of the actuation bar.

Step 3 In the next step, the second angle has to be found. First, the distance between the actuation base point and the tracked point on the puppet on a plane parallel to the $x-y$ plane is determined. This distance is given by

$$l_{act,arm,l,nearest} = \sqrt{(x_{arm,l} - t_{x,act,arm,l})^2 + (y_{arm,l} - t_{y,act,arm,l})^2} \tag{3.12}$$

If $l_{act,arm,l,nearets} > l_{act,arm}$, then $\alpha_{act,arm,l}$ will be simply set to 0. This is the case when the arm is out of reach of the actuation system.

In the other case, the angle will be calculated as

$$\alpha_{act,arm,l} = \arccos\left(\frac{l_{act,arm,l,nearest}}{l_{act,arm}}\right) \tag{3.13}$$

Step 4 As a last point, the final DOF, the string length, must be calculated. With the previously found rotational angles, the end point of the actuation bar is given by

$$\mathbf{r}_{W_O C_{arm,l,3}} = \begin{pmatrix} t_{x,act,arm,l} + l_{act,arm} \cdot \sin\gamma_{act,arm,l} \cdot \cos\alpha_{act,arm,l} \\ t_{y,act,arm,l} + l_{act,arm} \cdot \cos\gamma_{act,arm,l} \cdot \cos\alpha_{act,arm,l} \\ z_{body} + L_{shoulder} - h_{act,arm} - l_{act,arm} \cdot \sin\alpha_{act,arm,l} \end{pmatrix} \tag{3.14}$$

The length of the string can then be found as

$$L_{arm,l} = \left\| \mathbf{r}_{W_O C_{arm,l,3}} - \mathbf{r}_{W_O A_{l,s}} \right\| \tag{3.15}$$

3.3 User Interface

To run a performance on the system, a sophisticated graphical user interface as can be seen in Figure 3.6 was created. This interface also allows the user to view the simulation.

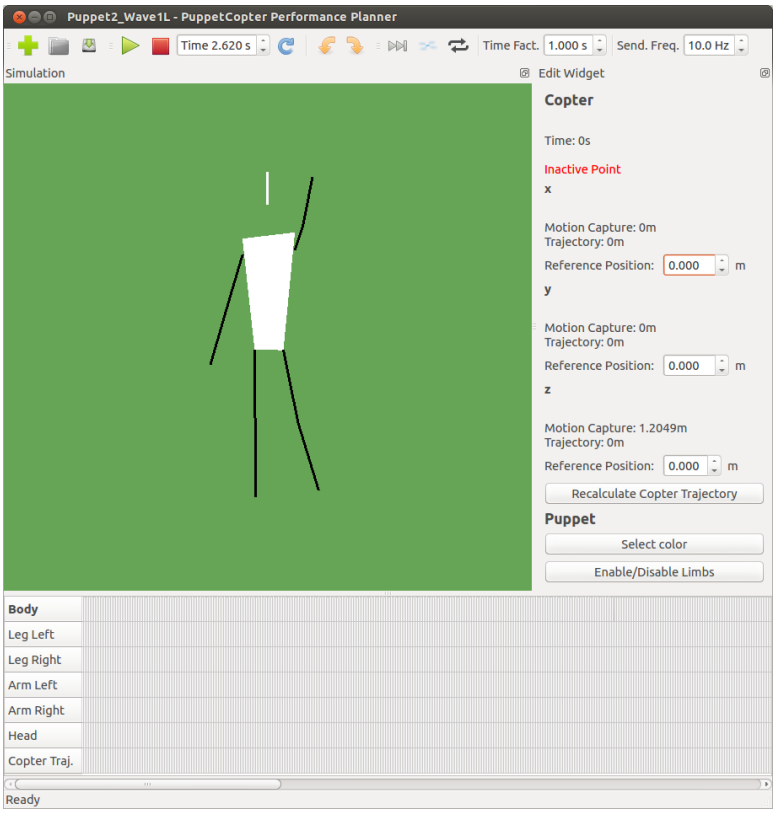


Figure 3.6: The graphical user interface. On top, there is a toolbar which provides access to the most important functions. Below that is the simulation on the left. On the right, there are options to define the copter trajectory and further tune the performance. In the bottom, there is a time line of the performance.

The user interface was created with the aid of Qt[2]. Qt is `C++` framework which allows for an easy creation of graphical user interfaces. There are various websites and books like [2] which explain the usage of this library.

In order to start the program, a `.pcr` file containing the performance has to be specified as a command line parameter. The specified file is the output of the MATLAB program. First, the simulation will be calculated if this has not been done before. Afterwards, the software is awaiting commands from the user.

There are four main areas. In the middle, there are two different parts. On the left, there is the simulation. The user can play the simulation and rotate the camera around to get a better view on the simulated puppet.

On the right, there is a functionality to edit the trajectory of the copter. This trajectory is very basic and only allows the user to create paths consisting of several straight lines. Due to the controller design (see [3] for details), it is not possible to directly use the recorded data as trajectory. The recorded data would have to be

[2]Website: https://qt-project.org/ (June 2013)

filtered and smoothed first in order to prevent misbehaviour of the copter. To help the user create a new path for the copter, the recorded data is shown as a reference.

Below the copter trajectory editor, there are two more buttons allowing to change settings of the puppet. The first button gives the user the opportunity to select a colour the puppet will light in. Behind the second button, there is a dialog which lets the user change the name of the performance. This dialog also includes the possibility to activate or deactivate certain limbs during the performance. If a limb is deactivated, the output for the actuation stays at a constant value. However, this functionality has not yet been thoroughly tested.

On the bottom of the window, there is a time line showing the whole performance. Here, the user can select a point and directly modify this point. On top of the window, there is a tool bar providing access to the most used features. These functions include:

- It is possible to extend the performance by adding another file to it. This extends the current performance. For the puppet movements, no special measurements are done when combining two performance files together. For the copter, the trajectory of the newly added data will be shifted such that the trajectory begins at the place where it ended in the original configuration.
- There is a button to run the performance on the system. There is an option to disable the copter which then only shows the puppet's motions. If the copter is enabled, a message will appear when the performance is started. This gives to user time to correctly initialize the copter. It is also possible to enable looping of the performance. Once activated, the performance will automatically start over after it has finished.
- For debugging purposes, it is possible to edit the frequency with which the performance data is sent. It is also possible to slow down the performance.

Chapter 4

Evaluation

In the following Chapter, the results of this Thesis are presented. The desired recorded movements from the motion capture are compared with the simulation results and the performance of the real puppet. The actual performance of the puppet was again recorded in the OptiTrack motion capture environment. There are a few general things that have to be noted first:

- The used coordinate frame has its origin in the middle of the shoulders (point B_0 in Figure 2.3). The coordinate axes point in the same direction as these of the inertial frame (I frame in Figure 2.2): The x axis points to the front of the puppet, the y axis goes to the left and the z axis goes upwards.
- The data from the puppet movement evaluation had to be shifted in time for a better comparison. The reason for doing this is that the beginning of the performance and the beginning of the recording do not coincide.
- The real system can do movements that are not possible in the simulation and are not desired to appear in the real system. These include for example a sideways motion (y direction) of the legs. These effects can easily appear with the real system since the puppet is very light and therefore massively influenced by all executed movements.
- There is a small offset of the shown positions. This is due to the fact that in the motion capture setup the marker ball was recorded and not the actual position of the string on the puppet limb. Additionally, the marker balls are not totally in the same place during the evaluation and the input recording.
- Another reason for the offsets in the beginning is that the puppet is not placed optimally in the beginning of the recording session.

With this information in mind, the evaluation of the results can be shown. First, the data of the walking movement and then the data from the waving animation is discussed.

4.1 Walking

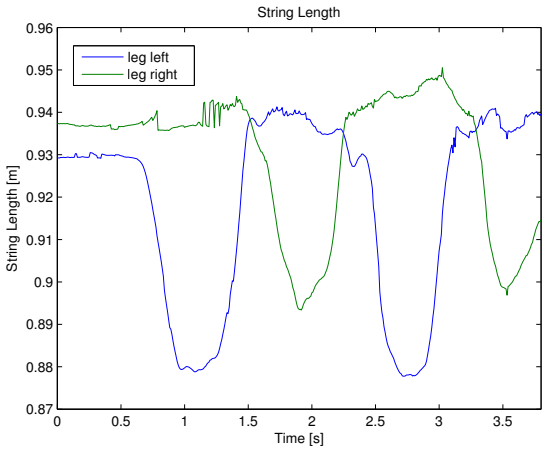


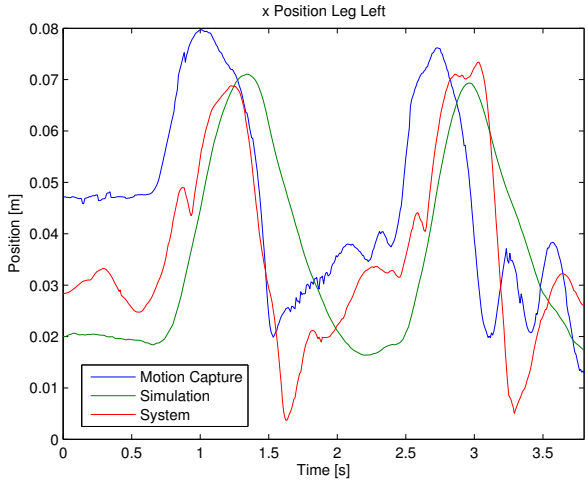
Figure 4.1: Calculated string lengths for the legs for the walking animation.

Figure 4.1 shows the calculated string lengths for the legs for the recorded walking movement. It can be nicely observed that alternatively the string length of the left and the right leg shrinks which results in an upwards movement of the leg. The offset in the initial position result from slightly different placed legs during the recording of this movement.

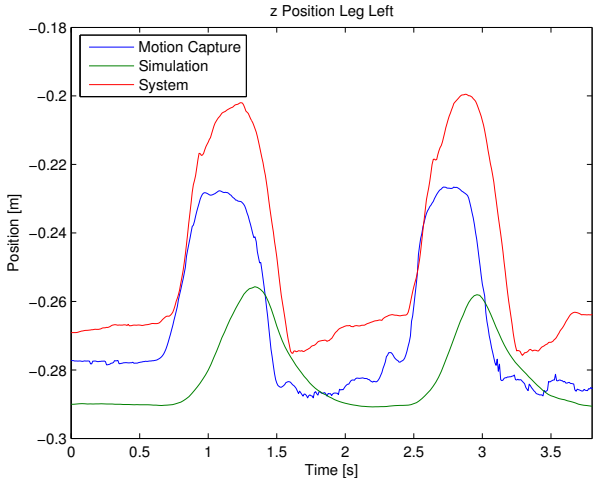
Figure 4.2 shows the position of the string on the left leg, Figure 4.3 gives the position of the string on the right leg in plot form. As can be seen in the plots, the simulation follows nicely the input data. The simulation data does not move as far as the real system which could be solved by further tweaking the simulation. However, the desired movement can be clearly seen in the simulation.

As for the real system, there was an overshoot on the left leg which can also be seen in the plots of the right leg (two peaks at time $t_1 = 1.2\,\mathrm{s}$ and $t_2 = 2.9\,\mathrm{s}$). This is also the reason for the two peaks at these times in the plot of the x coordinate of the right leg (Figure 4.3(a)). In the end, the walking movement of the puppet can still be recognized.

Figure 4.4 shows images of the recorded input as well as the simulation and the system behaviour.

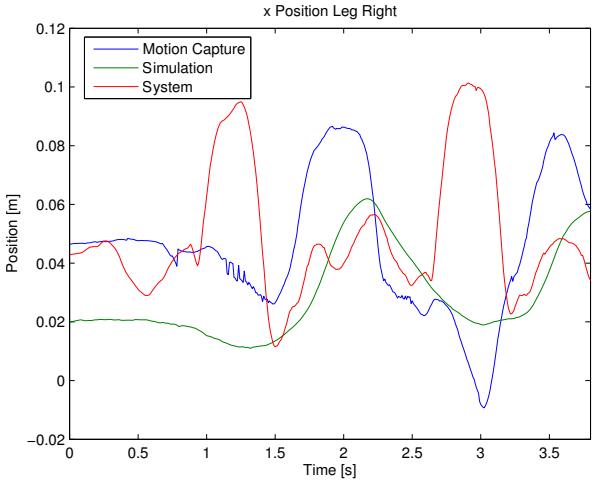


(a)

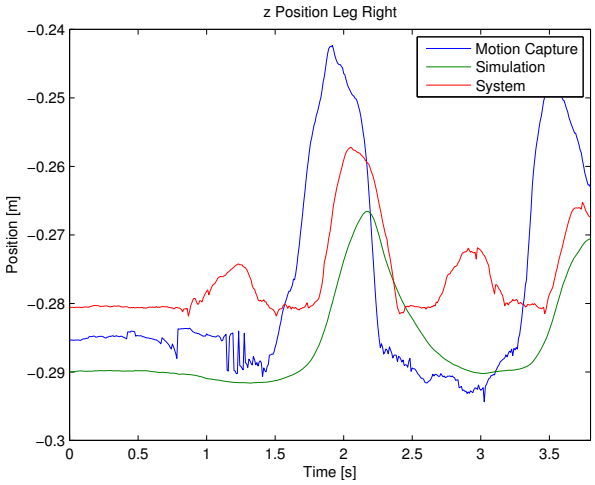


(b)

Figure 4.2: Comparison of the position of the string on the left leg between recorded data from motion capture (blue), simulation (green) and the real system (red). (a) shows the x coordinate, (b) shows the z coordinate.

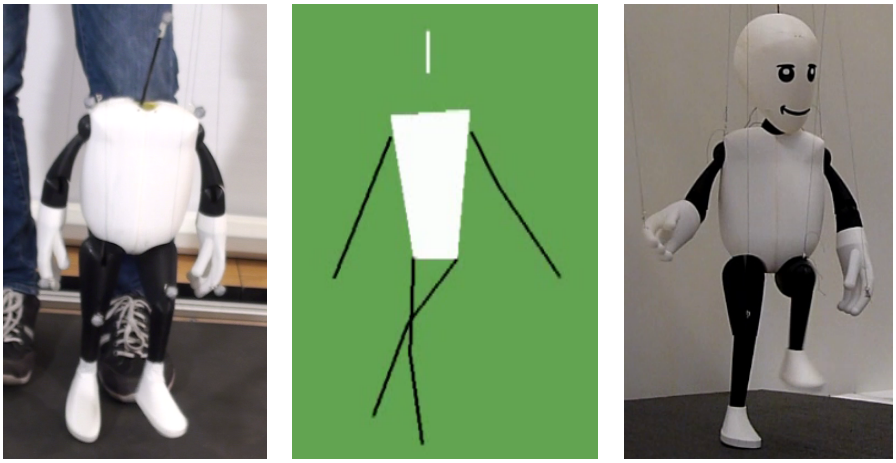


(a)



(b)

Figure 4.3: Comparison of the position of the string on the right leg between recorded data from motion capture (blue), simulation (green) and the real system (red). (a) shows the x coordinate, (b) shows the z coordinate.



(a) Motion capture (b) Simulation result (c) System behaviour

Figure 4.4: Comparison of the input with the system behaviour and the simulation of the walking motion.

4.2 Waving

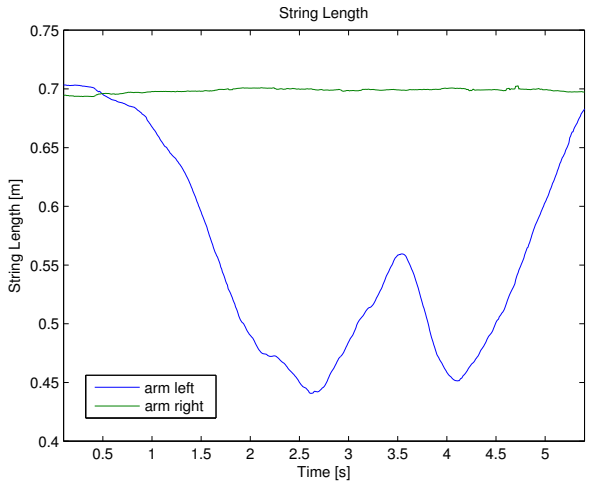


Figure 4.5: Calculated string lengths for the legs for the waving animation.

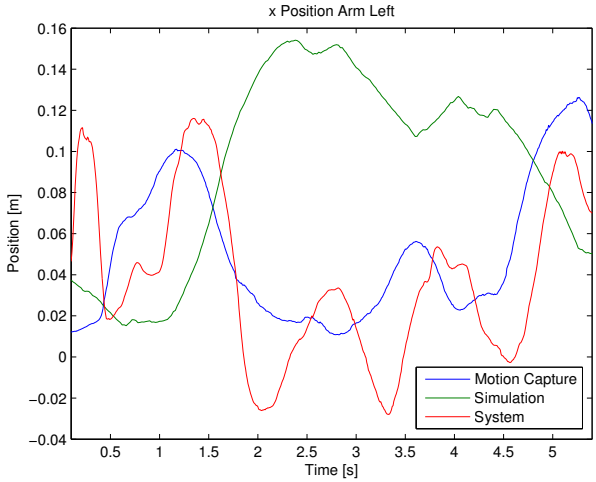
In Figure 4.5, the calculated string length of the arms for the waving animation can be seen. As expected, the string length of the right arm stays at a constant level since the input is to only move one arm, the left one. For the left arm, it can be seen how the string is moving up and down to perform the waving movement.

In Figure 4.6, the position of the string on the left arm can be seen. It can be seen that the simulation as well as the actual system follow nicely the input curve for the y and z coordinate. For the x coordinate, the motion capture input data and the system performance differ only slightly while the data from the simulation performs differently than the input. The main characteristics of the movement stay intact for the simulation, however, since the motions in x direction do not define the waving animation.

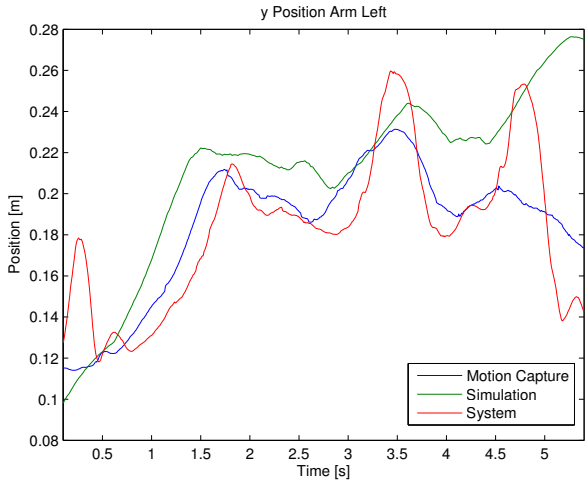
It can be also seen here that the simulation reacts a bit slow and with a small delay. No statement can be made about the delay behaviour of the real system since it difficult to determine the exact moment in time when the performance started.

The peak the system performs in the beginning is due to a different initialization. The system always starts with $\alpha_{act,arm,l} = 0°$ and $\gamma_{act,arm,l} = 0°$ while the commands directly start with a certain angle which is not necessarily zero.

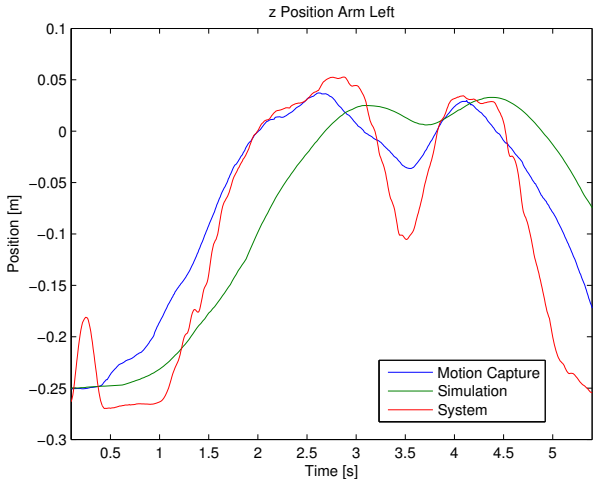
Figure 4.7 show images of the system behaviour and the simulation in comparison with the input from the motion capture recording.



(a)



(b)



(c)

Figure 4.6: Comparison of the position of the string on the left leg between recorded data from motion capture (blue), simulation (green) and the real system (red). (a) shows the x coordinate, (b) the y coordinate, and (c) shows the z coordinate.

(a) Motion capture (b) Simulation result (c) System behaviour

Figure 4.7: Comparison of the input with the system behaviour and the simulation of the waving motion.

Chapter 5

Conclusion

5.1 Summary

The tool developed in this bachelor thesis allows to now generate performances that can be played with the puppet. It is possible to record the desired movements by simply playing with the puppet used in the focus project. The simulation is able to show various movements which can be then by performed by the real system.

The user interface developed in this project is simple to use: It provides quick access to the functions of viewing the simulation and running the performance on the actual system. The good integration of the simulation in the user interface also supports the selected tool. Furthermore, the calculated actuation command of the software introduced in this report proofed to accurately reproduce the recorded movements of various different motions like walking, waving or flying.

5.2 Future Work

As with every project, there is always room for improvement. As a first step, it is suggested to incorporate some method to selectively enable or disable certain limbs during the performance. Additionally, further processing of the input data from the motion capture to filter out unwanted vibrations should be considered.

As another point, the actuation of the head can be implemented. For this, the corresponding actuation commands need to be calculated similarly to these of the arms and the simulation must be adapted to include the further string. Additionally, the head must also be actuated by the system.

Finally, there are several possibilities to enhance the user interface. For now, there is no direct open functionality in the software. Whenever another performance is started, the software needs to be closed first, and then restarted. Furthermore, the generation of a trajectory of the copter can be improved. For this, it is also necessary to consider [3].

Indices

Bibliography

[1] M. Ailinger, K. Egger, D. Freitag, R. Hofstetter, D. Keidel, P. Lustenberger, F. Martinoni, and G. Wiedebach. Focus Project PuppetCopter. Technical report, ETH Zurich, 2013.

[2] J. Blanchette and M. Summerfield. *C++ GUI programming with Qt 4*. Prentice Hall, 2006.

[3] F. Martinoni. Trajectory Control for a Multirotor Platform. Bachelor Thesis, ETH Zurich, 2013.

[4] M. Hutter and C. Gehring. *proNEu Documentation*. ETH Zurich, 2012.

[5] P. Lustenberger. Modular Actuators for Scalable Puppet Animation. Bachelor Thesis, ETH Zurich, 2013.

List of Figures

List of Tables

Appendix

Appendix A

Mechanical Properties of the Puppet

A.1 Actuation System

Variable	Matlab/C++	Value	Description
$h_{act,arm}$	`h_act_arm`	0.05 m	z offset between the actuation platform and actuation bar
$c_{cog,act,arm,y}$	`cog_actarm_x`	0.1 m	COG of the actuation bar
$l_{act,arm}$	`l_actarm`	0.2 m	length of the actuation bar
$t_{x,act,arm,l}$	`t_x_act_arm_l`	0 m	distance between base point of actuation unit and coordinate origin
$t_{y,act,arm,l}$	`t_y_act_arm_l`	0.0855 m	
$t_{x,act,arm,r}$	`t_x_act_arm_r`	0 m	
$t_{y,act,arm,r}$	`t_y_act_arm_r`	-0.0855 m	
$t_{x,act,leg,l}$	`t_x_act_leg_l`	0.09 m	
$t_{y,act,leg,l}$	`t_y_act_leg_l`	0.0825 m	
$t_{x,act,leg,r}$	`t_x_act_leg_r`	0.09 m	
$t_{y,act,leg,r}$	`t_y_act_leg_r`	-0.0825 m	
$t_{x,act,leg,back}$	`t_x_act_leg_back`	-0.122 m	
$t_{y,act,leg,back}$	`t_y_act_leg_back`	0 m	
$t_{x,act,head}$	`t_x_act_head`	0 m	
$t_{y,act,head}$	`t_y_act_head`	0 m	

A.2 Main Body

Variable	Matlab/C++	Value	Description
$L_{shoulder}$	`L_shoulder`	0.7 m	distance between shoulder and actuation platform
$c_{cog,body,x}$	`cog_body_x`	0.0044175 m	COG of the body expressed in the body's coordinate frame
$c_{cog,body,y}$	`cog_body_y`	0.00036212 m	
$c_{cog,body,z}$	`cog_body_z`	-0.092502 m	
m_{body}	`m_body`	0.15394 kg	mass of the body
$\theta_{xx,body}$	`Th_body_xx`	891.92 kg mm^2	inertias of the body expressed in the body's coordinate frame
$\theta_{yy,body}$	`Th_body_yy`	740.541 kg mm^2	
$\theta_{zz,body}$	`Th_body_zz`	426.385 kg mm^2	
$\theta_{xy,body}$	`Th_body_xy`	0.70102 kg mm^2	
$\theta_{xz,body}$	`Th_body_xz`	-26.183 kg mm^2	
$\theta_{yz,body}$	`Th_body_yz`	-4.3394 kg mm^2	

A.3 Upper Leg (Leg 1)

Variable	Matlab/C++	Value	Description
$t_{y,center,leg}$	t_y_center_leg	0.04 m	distance between the origin of the body (mid point of shoulder strings) and the origin of the upper leg (rotation point)
$t_{z,leg}$	t_z_leg	0.205 m	
$c_{cog,leg,1,x}$	cog_leg_1_x	0.00154555 m	COG of the upper leg expressed in the upper leg's coordinate frame
$c_{cog,leg,1,y}$	cog_leg_1_y	-0.0003402 m	
$c_{cog,leg,1,z}$	cog_leg_1_z	-0.022669 m	
$s_{leg,1,x}$	s_leg_1_x	0.02 m	front string location of the upper leg expressed in the upper leg's coordinate frame
$s_{leg,1,y}$	s_leg_1_y	0 m	
$s_{leg,1,z}$	s_leg_1_z	-0.095 m	
$s_{leg,back,x}$	s_leg_back_x	-0.02 m	back string location of the upper leg expressed in the upper leg's coordinate frame
$s_{leg,back,y}$	s_leg_back_y	0 m	
$s_{leg,back,z}$	s_leg_back_z	-0.095 m	
$l_{leg,1}$	l_leg_1	0.11 m	length of the limb
$l_{leg,1,0T}$	l_leg_1_0T	-0.025 m	offset of the coordinate origin and the upper physical end of the leg
$m_{leg,1}$	m_leg_1	0.01093 kg	mass of the upper leg
$\theta_{xx,leg,1}$	Th_leg_1_xx	15.9505 kg mm^2	
$\theta_{yy,leg,1}$	Th_leg_1_yy	15.5344 kg mm^2	inertias of the upper leg expressed in the upper leg's coordinate frame
$\theta_{zz,leg,1}$	Th_leg_1_zz	5.38076 kg mm^2	
$\theta_{xy,leg,1}$	Th_leg_1_xy	0.00561 kg mm^2	
$\theta_{xz,leg,1}$	Th_leg_1_xz	-0.3683 kg mm^2	
$\theta_{yz,leg,1}$	Th_leg_1_yz	-0.042 kg mm^2	

A.4 Lower Leg (Leg 2)

Variable	Matlab/C++	Value	Description
$c_{cog,leg,2,x}$	`cog_leg_2_x`	0.0213083 m	COG of the lower leg expressed in the lower leg's coordinate frame
$c_{cog,leg,2,y}$	`cog_leg_2_y`	0.00376667 m	
$c_{cog,leg,2,z}$	`cog_leg_2_z`	-0.082258 m	
$l_{leg,2}$	`l_leg_2`	0.135 m	length of the limb
$m_{leg,1}$	`m_leg_1`	0.01141 kg	mass of the lower leg
$\theta_{xx,leg,1}$	`Th_leg_1_xx`	27.4609 kg mm^2	inertias of the lower leg expressed in the lower leg's coordinate frame
$\theta_{yy,leg,1}$	`Th_leg_1_yy`	37.6423 kg mm^2	
$\theta_{zz,leg,1}$	`Th_leg_1_zz`	13.8006 kg mm^2	
$\theta_{xy,leg,1}$	`Th_leg_1_xy`	1.58943 kg mm^2	
$\theta_{xz,leg,1}$	`Th_leg_1_xz`	-9.5261 kg mm^2	
$\theta_{yz,leg,1}$	`Th_leg_1_yz`	-1.6611 kg mm^2	

A.5 Upper Arm (Arm 1)

Variable	Matlab/C++	Value	Description
$t_{y,center,arm}$	`t_y_center_arm`	0.073 m	distance between the origin of the body (mid point of shoulder strings) and the origin of the upper arm (rotation point)
$t_{z,arm}$	`t_z_arm`	0.03 m	
$c_{cog,arm,1,x}$	`cog_arm_1_x`	0.00000431 m	COG of the upper arm expressed in the upper arm's coordinate frame
$c_{cog,arm,1,y}$	`cog_arm_1_y`	0.0426084 m	
$c_{cog,arm,1,z}$	`cog_arm_1_z`	0.0043149 m	
$l_{arm,1}$	`l_arm_1`	0.085 m	length of the limb
$m_{arm,1}$	`m_arm_1`	0.00473 kg	mass of the upper arm
$\theta_{xx,arm,1}$	`Th_arm_1_xx`	6.36965 kg mm^2	inertias of the upper arm expressed in the upper arm's coordinate frame
$\theta_{yy,arm,1}$	`Th_arm_1_yy`	0.86444 kg mm^2	
$\theta_{zz,arm,1}$	`Th_arm_1_zz`	6.37976 kg mm^2	
$\theta_{xy,arm,1}$	`Th_arm_1_xy`	0.00079 kg mm^2	
$\theta_{xz,arm,1}$	`Th_arm_1_xz`	-0.0002 kg mm^2	
$\theta_{yz,arm,1}$	`Th_arm_1_yz`	0.38382 kg mm^2	

A.6 Lower Arm (Arm 2)

Variable	Matlab/C++	Value	Description
$c_{cog,arm,2,x}$	cog_arm_2_x	0.00173277 m	COG of the lower arm expressed in the lower arm's coordinate frame
$c_{cog,arm,2,y}$	cog_arm_2_y	0.0564517 m	
$c_{cog,arm,2,z}$	cog_arm_2_z	-0.0048561 m	
$s_{arm,2,x}$	s_arm_2_x	0.03 m	string location of the lower arm expressed in the lower arm's coordinate frame
$s_{arm,2,y}$	s_arm_2_y	0.13 m	
$s_{arm,2,z}$	s_arm_2_z	0 m	
$l_{arm,2}$	l_arm_2	0.15 m	length of the limb
$m_{arm,2}$	m_arm_2	0.01260 kg	mass of the lower arm
$\theta_{xx,arm,2}$	Th_arm_2_xx	23.7209 kg mm^2	inertias of the lower arm expressed in the lower arm's coordinate frame
$\theta_{yy,arm,2}$	Th_arm_2_yy	5.19533 kg mm^2	
$\theta_{zz,arm,2}$	Th_arm_2_zz	24.6714 kg mm^2	
$\theta_{xy,arm,2}$	Th_arm_2_xy	2.04588 kg mm^2	
$\theta_{xz,arm,2}$	Th_arm_2_xz	0.2063 kg mm^2	
$\theta_{yz,arm,2}$	Th_arm_2_yz	-5.8331 kg mm^2	

A.7 Head

Variable	Matlab/C++	Value	Description
$t_{z,head}$	t_z_head	0.055 m	z offset from the shoulder and the base point of the head (rotation centre)
$c_{cog,head,x}$	cog_head_x	0.0020461 m	COG of the head expressed in the head's coordinate frame
$c_{cog,head,y}$	cog_head_y	0.00382591 m	
$c_{cog,head,z}$	cog_head_z	0.0518943 m	
$s_{head,x}$	s_head_x	0.05 m	string location of the head expressed in the head's coordinate frame
$s_{head,y}$	s_head_y	0 m	
$s_{head,z}$	s_head_z	0.08 m	
$m_{leg,1}$	m_leg_1	0.02708 kg	mass of the head
$\theta_{xx,leg,1}$	Th_leg_1_xx	120.224 kg mm^2	inertias of the head expressed in the head's coordinate frame
$\theta_{yy,leg,1}$	Th_leg_1_yy	108.419 kg mm^2	
$\theta_{zz,leg,1}$	Th_leg_1_zz	117.478 kg mm^2	
$\theta_{xy,leg,1}$	Th_leg_1_xy	18.4732 kg mm^2	
$\theta_{xz,leg,1}$	Th_leg_1_xz	-22.623 kg mm^2	
$\theta_{yz,leg,1}$	Th_leg_1_yz	-11.022 kg mm^2	

Appendix B

Generated Files

This Section provides an overview of the contents of the various files generated throughout the software created in this report. All this files have their values separated by tab stop (`'\t'`).

B.1 Performance File

Table B.1 lists the content of the file generated from the MATLAB script. This file has the extension `.pcr`.

Column	Content	Unit
0	Time	s
1, 2, 3	x, y, z position of the copter recorded in the motion capture (for reference)	m
4	yaw angle of the copter (Ψ_{Copter})	rad
5, 6, 7	x, y, z position of the copter, describe the actual trajectory	m
8, 9, 10	x, y, z position of the puppet body	m
11	yaw angle of the puppet (Ψ)	rad
12, 13, 14	x, y, z position of the left leg	m
15, 16, 17	x, y, z position of the right leg	m
18, 19, 20	x, y, z position of the left arm	m
21, 22, 23	x, y, z position of the right arm	m
24, 25, 26	x, y, z position of the head	m
27	flag determining if performance is a flying movement	-
28	integer defining the colour the puppet should light in	-

Table B.1: The contents of each column of the output file generated by the MATLAB script.

B.2 Simulation Results

The simulation creates two output files, both in the folder

```
/home/(USER)/.puppetcopter/
```

The simulation results are saved under `(PERFORMANCENAME).pcs`. The content of this file is listed in Table B.2. Additionally, a log file is created during the simulation. This file is saved as `(PERFORMANCENAME).pcl` and has the content as in Table B.3.

Column	Content	Unit
0	Time	s
1-15	Values of generalized coordinates $\mathbf{q}$	rad

Table B.2: The contents of each column of the .pcs file generated as a result of the simulation.

Column	Content	Unit
0	Time	s
1-15	Values of $\tau(1)$ to $\tau(15)$: friction and damping in joints	kg/s^2
16-18	Force of the string on the front of the left leg $x/y/z$	N
19-21	Force of the string on the back of the left leg $x/y/z$	N
22-24	Force of the string on the front of the right leg $x/y/z$	N
25-27	Force of the string on the back of the right leg $x/y/z$	N
28-30	Force of the string on the left arm $x/y/z$	N
31-33	Force of the string on the right arm $x/y/z$	N
34-36	Force of the string on head $x/y/z$	N
37-39	Position of the string on the left leg	m
40-42	Position of the string on the right leg	m
43-45	Position of the string on the left arm	m
46-48	Position of the string on the right arm	m
49-52	Error of the string force controller left leg / right leg / left arm / right arm	m
53-67	Joint velocities $\dot{\mathbf{q}}$	rad/s
68-82	Joint accelerations $\ddot{\mathbf{q}}$	rad/s

Table B.3: The contents of each column of the pcl log file generated during of the simulation