University of Twente

EEMCS / Electrical Engineering
Control Engineering

Motion Control of a Humanoid Head

Ludo Visser

MSc report

Supervisors:
prof.dr.ir. S. Stramigioli
ir. E..C. Dertien
ir. G. van Oort
dr.ing. R. Carloni

June 2008

Report nr. 016CE2008
Control Engineering
EE-Math-CS
University of Twente
P.O.Box 217
7500 AE Enschede
The Netherlands

Abstract

This report describes the design and implementation of a motion control algorithm for a humanoid robotic head. The humanoid head consists of a neck with four degrees of freedom and two eyes (a stereo pair system) with one common and one independent degree of freedom. The kinematic and dynamic properties of the head are analyzed and modeled using bondgraphs and screw theory. A motion control algorithm is designed that receives, as input, the output of a vision processing algorithm and utilizes the redundancy of the joints. This algorithm is designed to enable the head to focus on and follow a target, showing human-like behavior. The dynamic model is used to analyze the performance of the control algorithm in a simulated environment. The algorithm has been implemented and tested on a real-time control platform. The migration from simulation environment to the real-time platform is governed by a step-by-step integration and testing procedure. After each step, the algorithm output is validated and its performance evaluated. The algorithm is implemented succesfully on a real-time PC-104 platform.

Preface

This report concludes my Master's program in Electrical Engineering. After three years of Bachelor studies, a year of Master's classes, a one-year internship in the USA and almost nine months working on this final assignment I have reached the end of my student life. I happily look back at a very educating, insightful, fun and at times stressful period of my life.

Especially these last couple of months at the Control Engineering group have been a wonderful experience for me. This assignment has been a challenge, but it was fun to (finally) apply what I learned and also learn new things. Completing the assignment is something I am proud of, but what would not have been possible without the help and support of many people. I would like to take this opportunity to thank them.

I owe many thanks to Stefano Stramigioli. Not only for teaching me and making me excited about robotics, but also for giving me the opportunity to do this assignment. I like to thank Gijs van Oort and Edwin Dertien for their advice and many tips during these last couple of months. I also wish to thank Raffaella Carloni, for reading, correcting and giving feedback on this report countless times, until I finally managed to do it right.

I thank Rob Reilink and Jan Bennik for doing this project together – I think we did a great job! It was fun working with you guys, even with Rob, despite his relentless commenting on my programming, soldering, `vi`, "WB", and other skills, or lack thereof.

I would also like to thank everbody at the CE group for the "gezelligheid". I will miss the sometimes lengthy but very useful coffee breaks.

Last but definitely not least, I would like to thank my family for their support throughout the years: in many ways, without you, I would not have been able to complete my studies.

Ludo
Enschede, June 2008

Contents

Abbreviations and Symbols

$\mathbf{a} \in \mathbb{R}^n$	Column vector of n elements
$\tilde{\mathbf{a}}$	Skew symmetric matrix form of $\mathbf{a}$
$\mathbf{A} \in \mathbb{R}^{n \times m}$	Matrix of n rows and m columns
$\mathcal{A}$	Vector space
$T_a\mathcal{A}$	Tangent space to $\mathcal{A}$
$\alpha, \theta, \ldots$	Angles
ω	Angular velocity
τ	Torque
$\mathbf{J}$	Jacobian matrix
Ψ_i	Coordinate system i
$\mathbf{p}^k$	A vector expressed in coordinate system Ψ_k
$\mathbf{v}^k$	Linear velocity of $\mathbf{p}^k$ expressed in Ψ_k
$\mathbf{p}_i^{k,j}$	A vector from the origin of Ψ_i to the origin of Ψ_j, expressed in coordinate system Ψ_k
$\mathbf{R}_i^j$	Rotation matrix that defines the rotation from Ψ_i to Ψ_j
$\mathbf{H}_i^j$	Homogeneous matrix that defines the change of coordinates from Ψ_i to Ψ_j
$\mathbf{T}_i^{k,j}$	Generalized velocities of coordinate system Ψ_i, with respect to coordinate system Ψ_j, expressed in coordinate system Ψ_k
$\hat{\mathbf{T}}$	Unit twist
$\mathbf{W}^{k,i}$	Generalized forces acting on body i, expressed in coordinate system Ψ_k
$\mathrm{Ad}_{\mathbf{H}_i^j}$	Adjoint of a homogeneous matrix
$\mathrm{ad}_{\mathbf{T}_i^{k,j}}$	Lie algebra of a twist
$\mathcal{I}^{k,i}$	Inertia tensor of body i, expressed in coordinate system Ψ_k
$\mathcal{P}^{k,i}$	Moment of body i, expressed in coordinate system Ψ_k
`Se`	An effort source in bondgraphs
`I`	An energy storing bondgraph element
`(M)TF`	A (modulated) transformer in bondgraphs
`(M)GY`	A (modulated) gyrator in bondgraphs
L,R	Associated with left or right camera/eye
proj	Projection, projected
GSL	GNU Scientific Library
SV	Singular Value

1 Introduction

In the last decades, robotics has been developed and applied in several fields, like automotive, packaging, transportation and other kinds of general factories in which automated systems can assist human beings. Slowly but steadily the next phase is entered: humanoid robots.

Since the world as it is today has been designed and built by humans, it is a logical choice to have robots that take the form of humans. In many scenarios, this requires less effort to get things done and getting places. Also, collaboration tasks involving both humans and robots become easier when robots are capable of doing the same things as humans.

The Control Engineering group at the University of Twente has extensive experience in the field of dynamic walkers. Recently the group has started to employ its knowledge in constructing a humanoid robot. In a collaboration project with the Technical Universities of Delft and Eindhoven, a teen-sized humanoid robot is under development and built with the aim to participate in the RoboCup soccer competition.

Within the scope of this project, a humanoid head has been developed. The head will serve two purposes. Firstly, it will become an integral part of the humanoid itself. Secondly, it will serve as a research platform for studies on human-machine interaction. It is believed that human-like behavior in humanoids is a key part in acceptance of humanoids in society.

The realization of the humanoid head, implies work on a mechanical design (both interior and exterior), a vision system and a motion control algorithm. The mechanical design consists of a four degree of freedom neck and the vision system of two cameras with two degrees of freedom. In particular, this research assignment focuses on the development on and implementation of the motion control algorithm.

1.1 Goal

The goal of this assignment is to develop and implement a motion control algorithm for the humanoid head. Inputs to the algorithm are the coordinates of a target, provided by the vision processing algorithm. The control algorithm should enable the head to look at things (even while moving), while at the same time the movements are human-like. The algorithm should thus be designed to exploit the redundancy of the system in order to generate human-like motions while performing the primary task of target tracking.

Since the development processes of the mechanical design, the vision processing algorithm and the motion control algorithm will run in parallel, an extensive dynamic model of the system should be developed in order to facilitate testing of the algorithm prior to implementation.

1.2 Outline

The project is divided into two parts, which have been documented in two separate papers. The first part covers the design and simulation of the motion control algorithm. This paper first treats the design of a dynamic model that is

used to test the algorithm in a simulation environment. Then the design of the motion control algorithm is covered. The kinematic behavior of the system is analyzed and a solution is presented that utilizes the redundancy in the system to meet the requirement of human-like motions.

The second part covers the implementation of the algorithm on a real-time platform. A step-wise testing and integration procedure is presented that helps migrating the scripted algorithm from the simulation environment to the real-time platform.

2 Design and Simulation

Vision Based Motion Control for a Humanoid Head

L.C. Visser, S. Stramigioli, R. Carloni, G. van Oort, E.C. Dertien

***Abstract*—This paper describes the design of a motion control algorithm for a humanoid robotic head. The humanoid head consists of a neck with four degrees of freedom and two eyes (a stereo pair system) with one common and one independent degree of freedom. The kinematic and dynamic properties of the head are analyzed and modeled using bondgraphs and screw theory. A motion control algorithm is designed as to receive, as an input, the output of a vision processing algorithm and to exploit the redundancy of the joints for the realization of the movements. This algorithm is designed to enable the head to focus on and to follow a target, showing human-like behavior. The dynamic model is used to analyze the performance of the control algorithm in a simulated environment.**

***Index Terms*—Bondgraphs, human-machine interaction, humanoids, motion control, redundancy, robot dynamics, robot kinematics, vision systems, screw theory**

I. Introduction

THE Control Engineering group at the University of Twente, in collaboration with the Technical Universities of Delft and Eindhoven and industry partners, are developing the 3TU humanoid robot "TUlip" [1].

In the scope of the project, a humanoid head system, equipped with a vision system and a neck, has been designed. The purpose of this work is to develop and implement a motion control algorithm for this system. In particular, input put to the algorithm are the coordinates of a generic target, provided by the vision processing system so that the head can track an object, and while moving, it exhibits a human-like behavior.

The paper is organized as follows: in Section II, human anatomical data are analyzed and a complete list of requirements for the system is presented. These requirements have been used in [2] for the design of the mechanical part of the head and in this paper for the design of the motion control algorithm. Section III presents the dynamic model of the system based on screw theory and bondgraphs and Section IV focuses on the description of the control of the humanoid head, based on the kinematic properties of the model. Finally, in Section V simulation results of the dynamic model are presented and discussed.

II. Requirements

The purpose of the humanoid head project is twofold. Firstly, it is meant to be mounted on the teen-sized humanoid robot "TUlip". As such, the head should enable the robot to perceive its environment and focus on specific targets. Secondly, the head will be used as research platform in the field of human-machine interaction. As such, the head system should be able to exhibit human-like behavior, e.g. observing the environment, focusing on particular features, and, maybe, interacting with people.

Since the head should be able to move human-like, it has been investigated what "human-like" actually means. Using research data from [3], [4] and observations, a set of anatomical data has been compiled: this list represents the average capabilities of a human head.

In particular, in order to achieve human-like motions, the head-neck system realizes a four degrees of freedom structure [2], as is shown in Fig. 1. The lower tilt (a rotation around the y-axis) and the pan (around the z-axis) motions of the head are realized through a differential drive, that combines the actuation of these two motions in a small area. The other two degrees of freedom of the neck are the roll (around the x-axis) motion and the upper tilt. The cameras mounted on a carrier structure share the common actuated tilt axis and can rotate side freely.

The specifications of the degrees of freedom for the neck and eyes are meant to approach the human anatomical data as closely as possible within reasonable feasibility boundaries. Since the head will be mounted on a humanoid robot, there are some severe restrictions on available space and power. The space limitations restrict the maximum angle certain joints can span, and power limitations restrict velocities and accelerations. Therefore, a trade-off was made between getting as close as possible to the human capabilities and the real feasibility. Table I summarizes the final set of specifications.

Aside from the mechanical requirements of the system, there are also behavioral requirements that the system should comply with in order to look "natural". Behavioral studies have shown how humans use both their head and eyes to look at a particular target [5]. In general the eyes move first towards the target, while the head slowly follows.

The direction of sight is called the gaze. The gaze is defined as the angle of the eyes with respect to a fixed reference and is equal to the sum of the angle of the head with respect to this reference and the angle of the eye with respect to the head. The offset of the eye with respect to the rotation point of the head is usually ignored. A gaze shift can be a saccade, when the gaze is abruptly changed (e.g. looking at a new object), or a smooth movement (e.g. following an object). Fig. 2 shows a simulated (one dimensional) saccade. It can be seen from this figure that the gaze (top) changes fast due to the fast movement of the eyes (middle). The head (bottom) moves slowly towards the target. When the eyes look at the target, they start to counter rotate to compensate for the movement of the head.

It is this kind of motions that characterizes humans: the fast and light-weight eyes acquire the target quickly, while the heavy head follows later and slower. This combines fast gaze shifts with low energy cost. This kind of motion should be mimicked by the motion control algorithm to make the system motions look human.

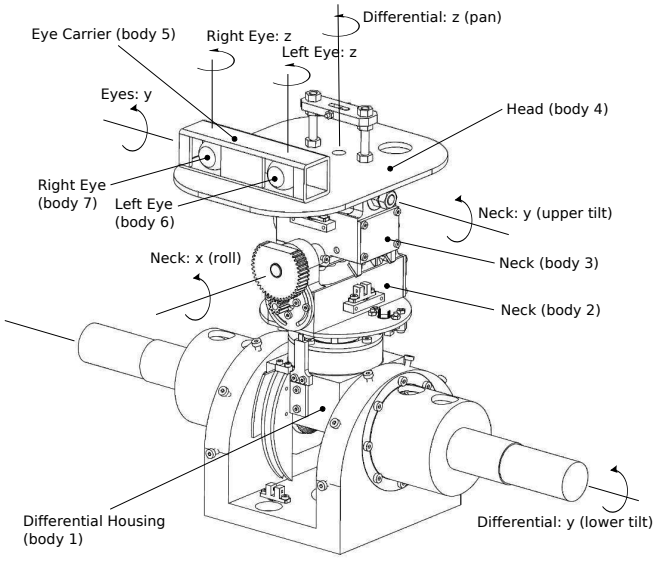

Fig. 1. Mechanical design of the neck-head system — The mechanical design comprises a four degree of freedom neck with a platform carrying the cameras (representing the eyes). The cameras tilt on a common axis, but rotate sideways independently. The differential combines the lower tilt and pan movement.

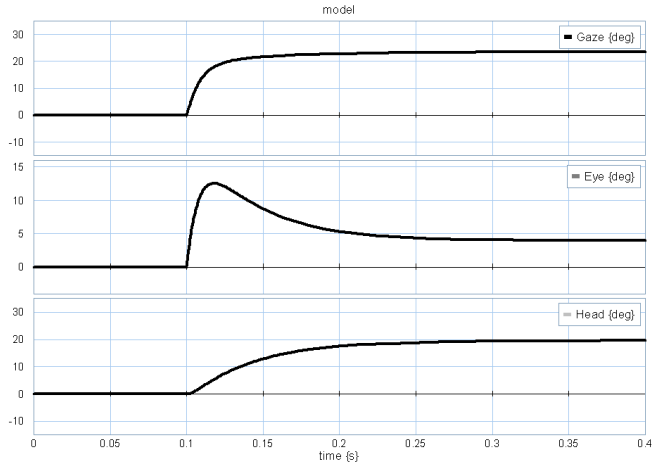

Fig. 2. A simulated saccade of a human — The gaze (top) is defined as the angle of the eyes with respect to a fixed reference. The sum of the angle of the eyes with respect to the head (middle) and the angle of the head with respect to the fixed reference (bottom) gives the gaze. The gaze quickly reaches the desired angle because of the fast movement of the eyes. The eyes keep the angle of the gaze constant by counter rotating to compensate for the relatively slow movement of the head.

TABLE I
FINAL SPECIFICATIONS COMPARED TO HUMAN ANATOMICAL DATA

		Human	**Model**
Head	Lower tilt	$-30^o - +10^o$	$-45^o - +45^o$
	Upper tilt	$-71^o - +100^o$	$-45^o - +45^o$
	Pan	$\pm 100^o$	$\pm 100^o$
	Roll	$\pm 63.5^o$	$\pm 35^o$
	v_{max}	$352^o/s$	$160^o/s$
	a_{max}	$3300^o/s^2$	$3300^o/s^2$
Eyes	Tilt	$\pm 40^o$	$\pm 30^o$
	Pan	$\pm 45^o$	$\pm 30^o$
	v_{max}	$850^o/s$	$600^o/s$
	a_{max}	$82000^o/s^2$	-
	Field of view (hor.)	175^o	60^o
	Field of view (ver.)	160^o	60^o
	Focal field of view	$\sim 2^o - 5^o$	-

III. DYNAMIC MODEL

The design of the head system consists of seven rigid bodies (a differential housing, two neck elements, the head, the eye carrier and two eyes), interconnected by joints. In order to facilitate algorithm development and testing, a dynamic model of this design has been developed using bondgraphs and screw theory. Bondgraph theory is well suited for fast and multi-domain dynamic modeling, while screw theory provides the mathematical tools to describe kinematic and dynamic relations of connected rigid bodies. The combination of these two theories provide a powerful and flexible toolset for dynamic modeling.

A. Rigid Bodies

In order to model the dynamic behavior of a generic rigid body, a number of essential properties of a rigid body need to be identified. With the aim of explaining the basis of screw theory, we refer to Fig. 3.

Each body i is characterized by a reference coordinate frame Ψ_i, centered in the joint connecting body i to a previous body $i-1$ and aligned with the joint rotation axis. This choice allows a easy modeling of a chain of rigid bodies.

The rigid body velocity of a coordinate frame can be expressed in the form of a twist, which takes the form of a six dimensional vector

$$\mathbf{T}_i^{k,j} = \begin{bmatrix} \omega \\ \mathbf{v} \end{bmatrix}, \tag{1}$$

where $\mathbf{T}_i^{k,j}$ denotes the generalized velocity of the body fixed in coordinate frame Ψ_i with respect to coordinate frame Ψ_j, expressed in Ψ_k. The twist has a rotational velocity component ω and a linear velocity component $\mathbf{v}$.

Secondly, a principal inertia frame, Ψ_{i_p}, is centered in the center of mass of the body. This coordinate frame is chosen such that it is aligned with the principal inertia axes of the body. By this choice, the inertia tensor of body i, denoted with $\mathcal{I}^i$, is diagonal when expressed in this frame, which greatly eases inserting this data or importing it from other software packages. Since the relative position of the main reference coordinate frame and the principal inertia frame is constant, the generalized velocity of these coordinate frames with respect to an arbitrary coordinate frame Ψ_j expressed in a global coordinate frame Ψ_0 are equal

$$\mathbf{T}_{i_p}^{0,j} = \mathbf{T}_{i_p}^{0,i} + \mathbf{T}_i^{0,j} = \mathbf{T}_i^{0,j}. \tag{2}$$

The impuls law for a point mass, $p = mv$, where p is the momentum of the point mass, m the mass and v the velocity, can be generalized to rigid bodies. The moment screw $\mathcal{P}^i$ of body i is given by [6]

$$\left(\mathcal{P}^i\right)^T = \mathcal{I}^i \mathbf{T}_i^{i,0}. \tag{3}$$

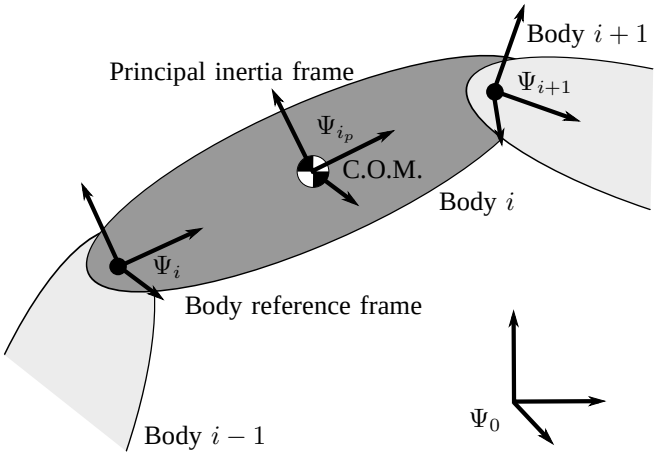

Fig. 3. Representation of a rigid body — The body coordinate frame Psi_i is chosen to be coincident with the joint connecting it to the previous body. A principal inertia coordinate frame is defined in the center of mass, aligned with the pricipal axes of the body. The inertia tensor of the body can be expressed in this frame in the form of a diagonal matrix.

The second law of dynamics for a rigid body, equivalent to $\dot{p} = F$, follows from Eq. (3)

$$\dot{\mathcal{P}}^{0,i} = \mathbf{W}^{0,i}, \tag{4}$$

where $\mathbf{W}^{0,i}$ represents the wrench acting on body i, expressed in the global coordinate frame Ψ_0. Eq. (4) can be expressed in the principal inertia frame of the body with a change of coordinates:

$$\left(\dot{\mathcal{P}}^{i_p}\right)^T = \mathrm{ad}^T_{\mathbf{T}^{i_p,0}_{i_p}} \left(\mathcal{P}^{i_p}\right)^T + \left(\mathbf{W}^{i_p}\right)^T, \tag{5}$$

where the symbol (ad) denotes the Lie algebra of $\mathbf{T}$.

The gravity force on body i is a wrench defined in a coordinate frame Ψ_{i_g} that is located in the principal inertia frame Ψ_{i_p}, but aligned with Ψ_0. By this choice, the wrench takes the intuitive form of

$$\mathbf{W}^{i_g}_{grav} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -mg \end{bmatrix}. \tag{6}$$

Bondgraph Representation: The generalized velocities of coordinate frames can be represented in a bondgraph structure by a `1`-junction. The inertia tensor $\mathcal{I}$ can be represented by an `I`-element. As stated before, the inertia tensor is diagonal if it is expressed in the principal inertia frame. Therefore, the `I`-element representing this tensor should be connected to a `1`-junction representing the generalized velocity of the principal inertia frame expressed in this frame, $\mathbf{T}^{i_p,j}_{i_p}$. This generalized velocity is related to $\mathbf{T}^{0,j}_{i_p}$ by the adjoint of the transformation matrix $\mathbf{H}$ that defines the change of coordinates

$$\mathbf{T}^{0,j}_{i} = \mathbf{T}^{0,j}_{i_p} = \mathrm{Ad}_{\mathbf{H}^0_{i_p}} \mathbf{T}^{i_p,j}_{i_p}. \tag{7}$$

This adjoint operation can be modeled by a `(M)TF`-element connecting the two `1`-junctions.

By representing the kinetic energy storage of the rigid body by an `I`:$\mathcal{I}^{i_p}$-element, the moment of body i, $\mathcal{P}^{i_p}$, is given by the state of the `I`-element, which can modulate a one-port `MGY`-element to implement the first term of the right hand side of Eq. (5).

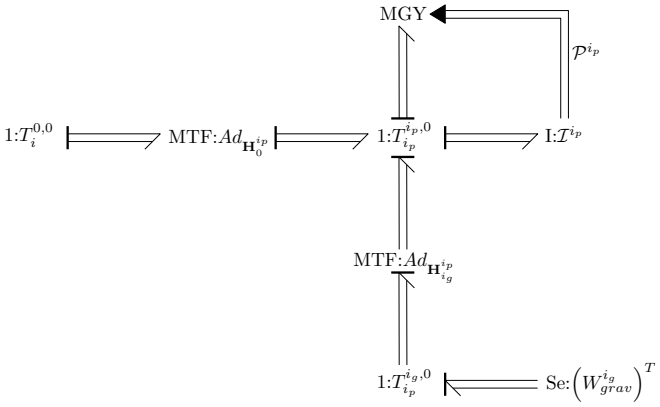

Fig. 4. Bondgraph representation of a rigid body — The generalized velocities of the two coordinate frames defined on the body are represented by `1`-junctions. Energy storage is modeled as a `I`-element.

By applying the proper change of coordinates to Eq. (6), gravity can be modeled as an `Se`-element in Ψ_{i_p}.

All components are now available to represent a rigid body in a bondgraph model, as shown in Fig. 4. Note that the bond to the `I`-element is of mixed causality. Only one state of the element is independent, the other five are dependent.

B. Joints

The rigid bodies of the system are interconnected by actuated joints. The torque τ and the rotational velocity ω of the output shaft of the joint motor are transformed into a twist and wrench pair that defines the joint motion. This transformation is defined by a Jacobian matrix $\mathbf{J}$ [6]

$$\begin{aligned} \mathbf{T} &= \mathbf{J}\omega \\ \mathbf{W} &= \tau \mathbf{J}^T, \end{aligned} \tag{8}$$

where ω and τ are in general scalars and $\mathbf{J}$ is a column vector equal to the unit twist $\hat{\mathbf{T}}$. For example, the Jacobian for the roll motion of the neck, i.e. a rotation about the local x-axis, would have:

$$\mathbf{J}_i = \hat{\mathbf{T}}^{i-1,i-1}_i = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T, \tag{9}$$

where the unit twist $\mathbf{T}^{i-1,i-1}_i$ gives the relative generalized velocity of the bodies connected by the joint. Since the body coordinate frames are chosen to be aligned with the joint rotation axis, the Jacobian in Eq. (9) takes the form of the unit twist with only one non-zero element, which gives an intuitive representation of the joint movement.

Eq. (9) holds for most of the joints, except for the differential drive. The twist of the body attached to the differential drive is a function of two actuators, hence its Jacobian $\mathbf{J}_{\text{diff}} \in \mathbb{R}^{6\times 2}$. In order to explain the derivation of the twist of the differential drive, we refer to the schematic representation depicted in Fig. 5.

The generalized velocity of frame Ψ_1, located in the center of the common gear, with respect to frame Ψ_0 as a function of ω_a and ω_b, the rotational velocities of the driven gears, can be found by considering the constraints imposed on the contact points $\mathbf{c}_1$ and $\mathbf{c}_2$.

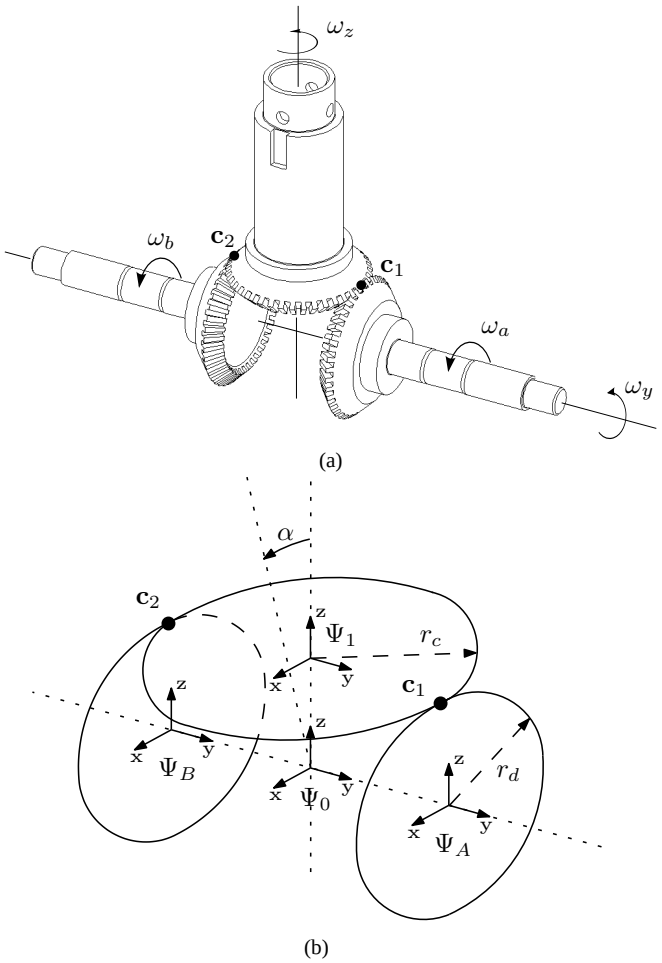

Fig. 5. Schematic representation of the differential drive — Fig. 5a shows a schematic drawing of the design, that shows how the motions of the common (upper) gear is constrained by the motion of the two driven gears. Fig. 5b shows a schematic representation of the differential drive that shows how the coordinate frames are defined. The constraint on the instantaneous linear velocity of the contact points $\mathbf{c}_1$ and $\mathbf{c}_2$, can be used to derive the twist of coordinate frame Ψ_1 with respect to global reference coordinate frame Ψ_0.

The contact points $\mathbf{c}_1$ and $\mathbf{c}_2$ can be expressed in homogeneous coordinates in Ψ_0 as

$$\mathbf{c}_1^0 = \begin{bmatrix} r_d \sin\alpha \\ r_c \\ r_d \cos\alpha \\ 1 \end{bmatrix}, \mathbf{c}_2^0 = \begin{bmatrix} r_d \sin\alpha \\ -r_c \\ r_d \cos\alpha \\ 1 \end{bmatrix}, \tag{10}$$

where the angle α is the angle of the z-axis of frame Ψ_1 with respect to the z-axis of frame Ψ_0 and r_c and r_d are the radii of the common and driven gears. From Fig. 5 it is straightforward that α is given by

$$\alpha = \frac{1}{2}\left(\theta_a + \theta_b\right), \tag{11}$$

where θ_a and θ_b denote the angle rotated by the driven gears, i.e. the integral of ω_a and ω_b respectively.

Let $\mathbf{p}_1$ be a point fixed in Ψ_1 and $\mathbf{p}_A$ be a point fixed in Ψ_A (on gear A). Furthermore, let both $\mathbf{p}_1$ and $\mathbf{p}_A$ be coincident with the contact point $\mathbf{c}_1$. The linear velocity of $\mathbf{p}_1$ and $\mathbf{p}_A$, numerically expressed in Ψ_0, must be equal when the gears are assumed to be ideal (i.e. no backlash) [7]. The linear velocity of $\mathbf{p}_1$ expressed in Ψ_0 is given by

$$\begin{aligned} \dot{\mathbf{p}}_1^0 &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\mathbf{H}_1^0 \mathbf{p}_1^1\right) \\ &= \dot{\mathbf{H}}_1^0 \mathbf{p}_1^1 + \mathbf{H}_1^0 \dot{\mathbf{p}}_1^1 = \dot{\mathbf{H}}_1^0 \mathbf{p}_1^1 \\ &= \dot{\mathbf{H}}_1^0 \left(\mathbf{H}_0^1 \mathbf{H}_1^0\right) \mathbf{p}_1^1 \\ &= \tilde{\mathbf{T}}_1^{0,0} \mathbf{H}_1^0 \mathbf{p}_1^1 = \tilde{\mathbf{T}}_1^{0,0} \mathbf{p}_1^0, \end{aligned} \tag{12}$$

where $\mathbf{H}_1^0$ is a homogeneous matrix that defines the change of coordinates from Ψ_1 to Ψ_0. A similar result is obtained for $\mathbf{p}_A$

$$\begin{aligned} \dot{\mathbf{p}}_A^0 &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\mathbf{H}_A^0 \mathbf{p}_A^A\right) = \dot{\mathbf{H}}_A^0 \mathbf{p}_A^A \\ &= \tilde{\mathbf{T}}_A^{0,0} \mathbf{H}_A^0 \mathbf{p}_A^A = \tilde{\mathbf{T}}_A^{0,0} \mathbf{p}_A^0. \end{aligned} \tag{13}$$

Since $\mathbf{p}_1$ and $\mathbf{p}_A$ are both coincident with $\mathbf{c}_1$, Eq. (12) and Eq. (13) must be equal

$$\tilde{\mathbf{T}}_1^{0,0} \mathbf{p}_1^0 = \tilde{\mathbf{T}}_A^{0,0} \mathbf{p}_A^0, \tag{14}$$

or equivalently

$$\tilde{\mathbf{T}}_1^{0,0} \mathbf{c}_1^0 = \tilde{\mathbf{T}}_A^{0,0} \mathbf{c}_1^0. \tag{15}$$

With an analogous approach for $\mathbf{c}_2$, the following set of equations is obtained

$$\begin{aligned} \tilde{\mathbf{T}}_1^{0,0} \mathbf{c}_1^0 &= \tilde{\mathbf{T}}_A^{0,0} \mathbf{c}_1^0 \\ \tilde{\mathbf{T}}_1^{0,0} \mathbf{c}_2^0 &= \tilde{\mathbf{T}}_B^{0,0} \mathbf{c}_2^0. \end{aligned} \tag{16}$$

The right hand sides of Eq. (16) are defined by Eq. (10) and

$$\tilde{\mathbf{T}}_A^{0,0} = \begin{bmatrix} 0 & 0 & \omega_a & 0 \\ 0 & 0 & 0 & 0 \\ -\omega_a & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tilde{\mathbf{T}}_B^{0,0} = \begin{bmatrix} 0 & 0 & \omega_b & 0 \\ 0 & 0 & 0 & 0 \\ -\omega_b & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{17}$$

The linear velocity of Ψ_1 expressed in Ψ_0 is zero by design, i.e.

$$\mathbf{T}_1^{0,0} = \begin{bmatrix} \omega^T & \mathbf{0} \end{bmatrix}^T. \tag{18}$$

By combining Eqs. (10), (16), (17), (18) it follows that

$$\mathbf{T}_1^{0,0} = \begin{bmatrix} \frac{1}{2}\frac{r_d}{r_c} \cdot \sin\alpha \cdot (\omega_b - \omega_a) \\ \frac{1}{2}\left(\omega_a + \omega_b\right) \\ \frac{1}{2}\frac{r_d}{r_c} \cdot \cos\alpha \cdot (\omega_b - \omega_a) \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{19}$$

which can be rewritten in the form of Eq. (8):

$$\begin{aligned} \mathbf{T}_1^{0,0} &= \mathbf{J}_{\text{diff}} \begin{bmatrix} \omega_a \\ \omega_b \end{bmatrix} \\ &= \begin{bmatrix} -\frac{1}{2}\frac{r_d}{r_c}\sin\alpha & \frac{1}{2}\frac{r_d}{r_c}\sin\alpha \\ \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2}\frac{r_d}{r_c}\cos\alpha & \frac{1}{2}\frac{r_d}{r_c}\cos\alpha \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \omega_a \\ \omega_b \end{bmatrix}. \end{aligned} \tag{20}$$

In the mechanical design, there is a differential housing present with a non-neglectable mass that only rotates along the y-axis of the differential joint. Therefore, $\mathbf{T}_1^{0,0}$ should be

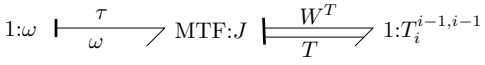

Fig. 6. Bondgraph representation of a joint — The (M)TF-element defines the relation between actuator output (ω, τ) and the movement of the body connected to the actuator.

decoupled in two separate rotations along the y- and z-axes. It can be shown that Eq. (20) can be decoupled as:

$$\begin{aligned}\mathbf{T}_1^{0,0} &= \mathbf{J}_{\text{decoupled}} \begin{bmatrix}\omega_y \\ \omega_z\end{bmatrix} \\ &= \begin{bmatrix}0 & \sin\alpha \\ 1 & 0 \\ 0 & \cos\alpha \\ 0 & 0 \\ 0 & 0 \\ 0 & 0\end{bmatrix} \begin{bmatrix}\frac{1}{2}(\omega_a + \omega_b) \\ \frac{1}{2}\frac{r_d}{r_c}(\omega_b - \omega_a)\end{bmatrix}.\end{aligned} \tag{21}$$

Bondgraph Representation: A joint defined by Eq. (8) can be represented in a bondgraph by an (M)TF-element, as shown in Fig. 6. In general, the left bond is only one-dimensional, as follows from Eq. (8). In case of the differential joint, the bond is two-dimensional, as follows from Eq. (20). It should be noted that the (M)TF-element has a fixed flow in causality, due to the non-invertability of Eq. (8).

C. Connecting Bodies and Joints

The generalized velocity of body i with respect to Ψ_0 expressed in Ψ_0, $\mathbf{T}_i^{0,0}$ is given by adding the generalized velocity of the previous body $i-1$, $\mathbf{T}_{i-1}^{0,0}$ and the relative velocity $\mathbf{T}_i^{0,i-1}$

$$\mathbf{T}_n^{0,0} = \mathbf{T}_1^{0,0} + \mathbf{T}_2^{0,1} + \cdots + \mathbf{T}_{n-1}^{0,n-2} + \mathbf{T}_n^{0,n-1}. \tag{22}$$

Using this relation, a kinematic chain of rigid bodies and joints can be represented in a bondgraph structure.

By properly transforming the generalized velocities of the main coordinate systems to a global coordinate frame Ψ_0, the joints and rigid bodies can be interconnected. This transformation can be implemented by an (M)TF-element using the Adjoint of the transformation matrix $\mathbf{H}$ as defined in Eq. (7). Addition of the generalized velocities is implemented through 0-junctions. The resulting structure is shown in Fig. 7.

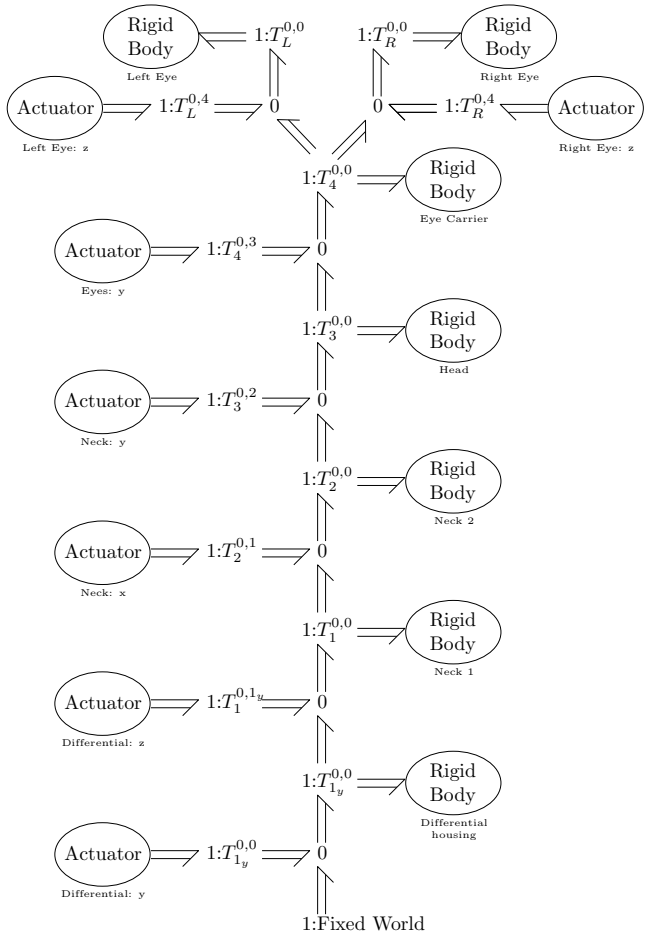

Fig. 7. Complete bondgraph structure — Joints and bodies can be connected by properly transforming generalized velocities to a common coordinate frame, after which they can be added using 0-junctions.

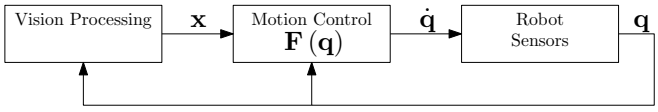

Fig. 8. Controller overview — The vision algorithm provides the motion control algorithm with target coordinates. From this, the controller calculates joint velocities through a map $\mathbf{F}(\mathbf{q})$. The change in joint state influences the target perception, so in order to come to a functional control algorithm, we need to know how the joints should be actuated and how this actuation influences the perception of the target.

IV. Motion Control

An overview of the controller structure is shown in Fig. 8. The vision processing algorithm determines where the robot head should look at by choosing a proper target in the image plane $\mathcal{X}$ [8]. The output of this algorithm, two sets of (x, y)-coordinates of the target, is supplied as input to the motion control algorithm. From these coordinates, the motion control algorithm calculates generalized joint velocities $\dot{\mathbf{q}}$ through the relation

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{q})\,\dot{\mathbf{q}}, \tag{23}$$

where $\dot{\mathbf{x}} \in T_x\mathcal{X}$, the tangent space to $\mathcal{X}$, denotes the time derivative of vector $\mathbf{x}$ of the target coordinates, $\mathbf{q} \in \mathcal{Q}$ denotes the generalized joint states defined in the vector space $\mathcal{Q}$ and $\dot{\mathbf{q}} \in T_q\mathcal{Q}$, the tangent space to $\mathcal{Q}$, its time derivative. It can be seen from the figure that the robot behavior influences the vision processing algorithm. In order to design an algorithm that can use the output of this algorithm effectively, two questions need to be answered: how is the target perceived by the camera and how does the joint actuation influence this perception. From this analysis, it is possible to derive a control law that actuates the joints so that the desired goal, i.e. looking at the target in a human-like way, is achieved.

A. Target Perception

Target perception by the camera can be modeled with a pinhole camera model [9]. From this model it follows that the coordinates come from a projection of the target on a image plane in the camera coordinate frame, as is shown in Fig. 9.

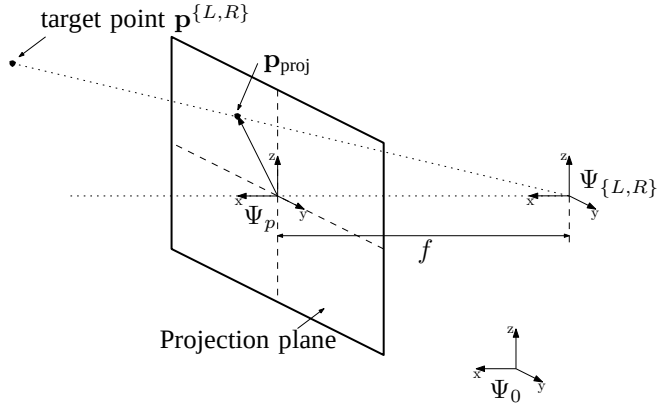

Fig. 9. Target coordinates — The target coordinates as perceived by the cameras can be modeled by a projection on a plane (the image plane) using a pinhole camera model. The camera coordinate frame is denoted by $\Psi_{\{L,R\}}$ for the left and right camera respectively. The projection plane is at focal depth f on the x-axis of the camera frame.

Let

$$\mathbf{p}^{\{L,R\}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}^{\{L,R\}} \tag{24}$$

where L and R identify the left and right cameras, be a target point in three dimensional Euclidian space $\mathcal{E}(3)$, expressed in coordinate frame $\Psi_{\{L,R\}}$. It can be shown that the projection of this target point, expressed in the camera coordinate frame, $\mathbf{p}_{\text{proj}}^{\{L,R\}}$, is given by

$$\mathbf{p}_{\text{proj}}^{\{L,R\}} = \begin{bmatrix} y_{\text{proj}} \\ z_{\text{proj}} \end{bmatrix}^{\{L,R\}} = \frac{f}{x^{\{L,R\}}} \begin{bmatrix} y \\ z \end{bmatrix}^{\{L,R\}}, \tag{25}$$

where f is the focal depth of the camera.

Assuming that the origin of the camera coordinate frame is located in the center of the image, "look at the target" is to be interpreted as $\mathbf{p}_{\text{proj}}$ being $(0,0)$ for both cameras. From this, the definition of the state vector $\mathbf{x}$ is formed to hold the projected target coordinates for both cameras:

$$\mathbf{x} = \begin{bmatrix} y_{\text{left}} \\ z_{\text{left}} \\ y_{\text{right}} \\ z_{\text{right}} \end{bmatrix} = \begin{bmatrix} \mathbf{p}_{\text{proj}}^{\text{L}} \\ \mathbf{p}_{\text{proj}}^{\text{R}} \end{bmatrix}. \tag{26}$$

B. Target Perception and Joint Movement

In order to move the robot to accomplish the required task, it is necessary to know how the state vector $\mathbf{x}$ changes as function of the change of the generalized joint coordinates $\mathbf{q}$, i.e. what the matrix $\mathbf{F}$ is in Eq. (23).

The generalized joint velocities in $\dot{\mathbf{q}}$ are given by the angular velocities of the joints:

$$\dot{\mathbf{q}} = \begin{bmatrix} \omega_y \\ \omega_z \\ \omega_{\text{neck,roll}} \\ \omega_{\text{neck,tilt}} \\ \omega_{\text{eyes,y}} \\ \omega_{\text{eye,left,z}} \\ \omega_{\text{eye,right,z}} \end{bmatrix}. \tag{27}$$

Using ω_y and ω_z is a more natural choice than using ω_a and ω_b from the actuators (see Fig 5). This requires that calculated generalized joint velocities as defined in Eq (27) need to be mapped to actuator velocities using the second part of Eq. (21).

The first step in finding $\mathbf{F}$ is to determine how the camera coordinate frames move as function of $\dot{\mathbf{q}}$. If $\mathbf{T}_{\{L,R\}}^{0,0}$ denotes the generalized velocity of the left (Ψ_L) or right (Ψ_R) camera coordinate frame with respect to the global coordinate frame Ψ_0, expressed in Ψ_0, it can be shown that the following relation holds [6]

$$\mathbf{T}_{\{L,R\}}^{0,0} = \mathbf{J}_{\{L,R\}}(\mathbf{q})\,\dot{\mathbf{q}} \qquad \mathbf{J}_{\{L,R\}} \in \mathbb{R}^{6\times 7}, \mathbf{q} \in \mathbb{R}^7, \tag{28}$$

where the Jacobian matrix $\mathbf{J}_{\{L,R\}}$ is constructed by the (unit) twists for each joint, as in Eqs. (9), (20). By evaluating Eq. (28), we can obtain an expression for the twists of the two cameras with respect to Ψ_0 in the same fashion as in Eq. (22).

From Eq. (28) it is found that the Jacobian for the left camera is given by

$$\mathbf{J}_L(\mathbf{q}) = \begin{bmatrix} \mathbf{J}_{\text{decoupled}} & \hat{\mathbf{T}}_2^{0,1} & \hat{\mathbf{T}}_3^{0,2} & \hat{\mathbf{T}}_4^{0,3} & \hat{\mathbf{T}}_L^{0,4} & \mathbf{0} \end{bmatrix}, \tag{29}$$

and, similarly, for the right camera

$$\mathbf{J}_R(\mathbf{q}) = \begin{bmatrix} \mathbf{J}_{\text{decoupled}} & \hat{\mathbf{T}}_2^{0,1} & \hat{\mathbf{T}}_3^{0,2} & \hat{\mathbf{T}}_4^{0,3} & \mathbf{0} & \hat{\mathbf{T}}_R^{0,4} \end{bmatrix}. \tag{30}$$

In Eqs. (29), (30) the twists $\hat{\mathbf{T}}_{\bullet}^{0,*}$ denote the unit twists as given in Eq. (9), but expressed in Ψ_0. The indices refer to the bodies, as defined in Fig. 1.

From Eq. (25) it follows that when the camera coordinate frame moves, the projection is affected, because $\mathbf{p}^{\{L,R\}}$ changes. An expression for the instantaneous rate of change of $\mathbf{p}^{\{L,R\}}$, $\dot{\mathbf{p}}^{\{L,R\}}$, caused by the joint movement, can be found by assuming a situation as depicted in Fig. 9 for the left camera.

Let the homogeneous coordinates of the target, expressed in the left camera coordinate frame Ψ_L, be given by

$$\begin{bmatrix} \mathbf{p}^L \\ 1 \end{bmatrix} = \mathbf{H}_0^L \begin{bmatrix} \mathbf{p}^0 \\ 1 \end{bmatrix}, \tag{31}$$

where $\mathbf{p}^0$ denotes the target coordinates in Ψ_0.

The linear velocity of $\mathbf{p}^L$ expressed in Ψ_L is found by differentiating Eq. (31) with respect to time, yielding

$$\begin{bmatrix} \dot{\mathbf{p}}^L \\ 0 \end{bmatrix} = \dot{\mathbf{H}}_0^L \begin{bmatrix} \mathbf{p}^0 \\ 1 \end{bmatrix}, \tag{32}$$

where we used the fact that we are considering the instantaneous case where $\dot{\mathbf{p}}^0 = 0$. By using the relation

$$\dot{\mathbf{H}}_0^L = \tilde{\mathbf{T}}_0^{L,L} \mathbf{H}_0^L, \tag{33}$$

we obtain

$$\begin{aligned} \begin{bmatrix} \dot{\mathbf{p}}^L \\ 0 \end{bmatrix} &= \tilde{\mathbf{T}}_0^{L,L} \mathbf{H}_0^L \begin{bmatrix} \mathbf{p}^0 \\ 1 \end{bmatrix} \\ &= \tilde{\mathbf{T}}_0^{L,L} \begin{bmatrix} \mathbf{p}^L \\ 1 \end{bmatrix}, \end{aligned} \tag{34}$$

that can be also written as

$$\dot{\mathbf{p}}^L = \begin{bmatrix} -\tilde{\mathbf{p}}^L & \mathbf{I}_3 \end{bmatrix} \mathbf{T}_0^{L,L}, \tag{35}$$

by using the relation $\tilde{\mathbf{a}}\mathbf{b} = -\tilde{\mathbf{b}}\mathbf{a}$. The twist $\mathbf{T}_0^{L,L}$ can be also written as

$$\mathbf{T}_0^{L,L} = -\mathrm{Ad}_{\mathbf{H}_0^L}\mathbf{T}_L^{0,0}, \tag{36}$$

by noting that $\mathbf{T}_j^{i,i} = -\mathbf{T}_i^{i,j}$ and $\mathbf{T}_i^{i,j} = \mathrm{Ad}_{\mathbf{H}_j^i}\mathbf{T}_i^{j,j}$. Finally, from Eq. (28), it follows that

$$\dot{\mathbf{p}}^L = \begin{bmatrix}\tilde{\mathbf{p}}^L & -\mathbf{I}_3\end{bmatrix}\mathrm{Ad}_{\mathbf{H}_0^L}\mathbf{J}_L(\mathbf{q})\,\dot{\mathbf{q}}. \tag{37}$$

From Eq. (25) $\mathbf{p}_{\text{proj}}^L$ is found to be a scaled version of $\mathbf{p}^L$, and therefore

$$\dot{\mathbf{p}}_{\text{proj}}^L = -\begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{p}}_{\text{proj}}^L & -\mathbf{I}_3\end{bmatrix}\mathrm{Ad}_{\mathbf{H}_0^L}\mathbf{J}_L(\mathbf{q})\,\dot{\mathbf{q}}, \tag{38}$$

where $\dot{\mathbf{p}}_{\text{proj}}^L$ denotes the two dimensional velocity vector that gives the instantaneous velocity of the observed target on the image plane and the projected target $\mathbf{p}_{\text{proj}}^L$ is given by Eq. 24.

By taking the same approach, we find a similar expression for the right camera, and these results combined gives the matrix $\mathbf{F}$ in Eq. (23).

C. Control Law

Now that it is known how the perception of the target changes as the joint angles change, a control law can be formulated for the actuation of the joints. The goal is to move the perceived target coordinates $\mathbf{p}_{\text{proj}}^{\{L,R\}}$ to $(0,0)$. The error in the state vector is defined as

$$\mathbf{x}_{\text{error}} = \begin{bmatrix}\mathbf{0}\\ \mathbf{0}\end{bmatrix} - \begin{bmatrix}\mathbf{p}_{\text{proj}}^L\\ \mathbf{p}_{\text{proj}}^R\end{bmatrix} = -\begin{bmatrix}\mathbf{p}_{\text{proj}}^L\\ \mathbf{p}_{\text{proj}}^R\end{bmatrix}. \tag{39}$$

Using a proportional gain K_p on this error, the desired rate of change of the state vector, $\dot{\mathbf{x}}$, is formulated as

$$\dot{\mathbf{x}} = -K_p\begin{bmatrix}\mathbf{p}_{\text{proj}}^L\\ \mathbf{p}_{\text{proj}}^R\end{bmatrix}. \tag{40}$$

To achieve the desired $\dot{\mathbf{x}}$, it is required to invert Eq. (23) to calculate the required joint velocities $\dot{\mathbf{q}}$. Since the robot head has multiple redundant joints, inversion of Eq. (23) is given by

$$\dot{\mathbf{q}} = \mathbf{F}^\sharp\dot{\mathbf{x}} + \left(\mathbf{I}_7 - \mathbf{F}^\sharp\mathbf{F}\right)\mathbf{z}, \tag{41}$$

where $\mathbf{F}^\sharp \in \mathbb{R}^{7\times 4}$ is the weighted generalized inverse of matrix $\mathbf{F}$ in Eq. (23) and vector $\mathbf{z} \in \mathbb{R}^7$ is an arbitrary vector of appropriate dimension which is projected onto the null space of $\mathbf{F}$, see [10] for more details[1].

In order to achieve the required human-like motions, we choose a criterion $\mathbf{G}(\mathbf{q})$ with respect to $\mathbf{q}$

$$\mathbf{G}(\mathbf{q}) = \frac{1}{2}(\mathbf{q}_0 - \mathbf{q})^T\mathbf{W}(\mathbf{q}_0 - \mathbf{q}), \tag{42}$$

[1]In general, $\dot{\mathbf{x}} = \mathbf{F}\dot{\mathbf{q}}$ has a minimum norm, least-squares solution $\dot{\mathbf{q}} = \mathbf{F}^\sharp\dot{\mathbf{x}}$ when a physically consistent Euclidian inner product is defined on both the vector space $T_x\mathcal{X}$ and $T_q\mathcal{Q}$. In this paper, since any element belonging to these vector spaces can be represented by components with the same physical units, a physically consistent inner product is defined and it follows that $\mathbf{F}^\sharp = \mathbf{F}^+ = \mathbf{F}^T\left(\mathbf{F}\mathbf{F}^T\right)^{-1}$, where the symbol $+$ indicates Moore-Penrose pseudoinverse.

where $\mathbf{q}_0$ denotes the preferred joint state (which may be chosen to be 0) and $\mathbf{W} = \mathrm{diag}\{w_1,\ldots,w_7\}$, where w_i are weighting factors. By taking

$$\mathbf{z} = \frac{\partial\mathbf{G}}{\partial\mathbf{q}} = \mathbf{W}(\mathbf{q}_0 - \mathbf{q}), \tag{43}$$

the joints are actuated to move towards their initial position $\mathbf{q}_0$ proportionally to the weighting factors w_i. By choosing appropriate weighting factors w_i, the head will move in a human-like way [11]. For example, giving the weight corresponding to the roll motion of the neck a large value will result in only small roll motions. This is desirable behavior, because humans do not often use this motion either. Also, the weights corresponding to the joints that control the eye motion need to have a significant value, so that the eyes tend to return to a neutral position, but at the same time a value that is not too high, to allow the eyes to move to the target in the first place. Choosing the exact values for w_i has turned out to be a combination of the above reasoning and trial-and-error.

A more sophisticated approach could be to choose the weights w_i dynamically, so that we have

$$\mathbf{W} = \mathbf{W}(t) = \mathrm{diag}\{w_1(t),\ldots,w_7(t)\}. \tag{44}$$

With this approach, the joint motion can be controlled over time to achieve more sophisticated behavior. For example, when tracking a slow moving target, the weights that control the eye motion could be choosen very low when a target has just come into view. As the target remains in view over time, the weights could increase in value, so that the eyes will remain closer to their neutral position and as a result the neck joints become more active. This kind of behavior closely resembles human behavior.

Optionally, the vector $\mathbf{z}$ can be used to project "behavioral motions" onto the null-space and in this way have the head show emotions while simultaneously looking at something, e.g. somebody's face.

V. Simulation Results

The dynamic model and the motion control algorithm have been implemented in a simulation environment using 20-sim simulation software [12]. This software package allows for direct implementation of bondgraph models combined with the controller code.

The motion control algorithm and the model are implemented in a structure depicted in Fig. 10. The cameras are also modeled using the pinhole camera model given by Eq. (25). The delay due to the time that the vision processing algorithm needs to process the camera images is also modeled.

The simulation environment is divided in a continuous time and discrete time part. It is expected that the vision processing algorithm will work at a rate of 20 Hz., therefore the cameras are modeled in discrete time at 20 Hz. The motion control algorithm is also modeled in discrete time, but at a rate of 60 Hz. It has been found that this is the minimum sample rate at which the motion control is capable of effectively dealing with the delay of the vision processing algorithm.

The joint velocities calculated by the motion control algorithm are send to the actuators of dynamic bondgraph model

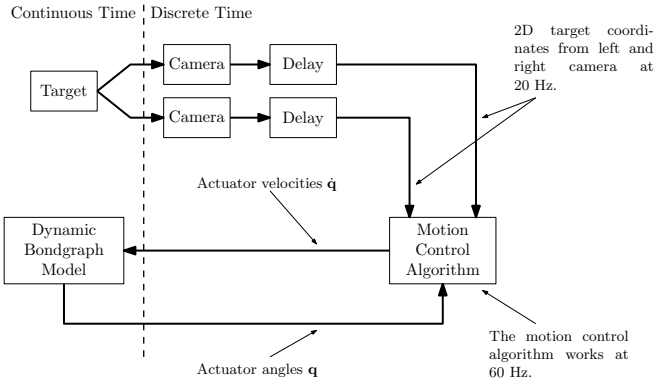

Fig. 10. Simulation environment in 20-sim — The target and the dynamic model are modeled in continuous time; the cameras and the motion control algorithm are modeled in discrete time. The cameras supply the algorithm with target coordinates at a fixed rate of 20 Hz. The dynamic model of the head also contains closed loop PID-controllers for the actuators.

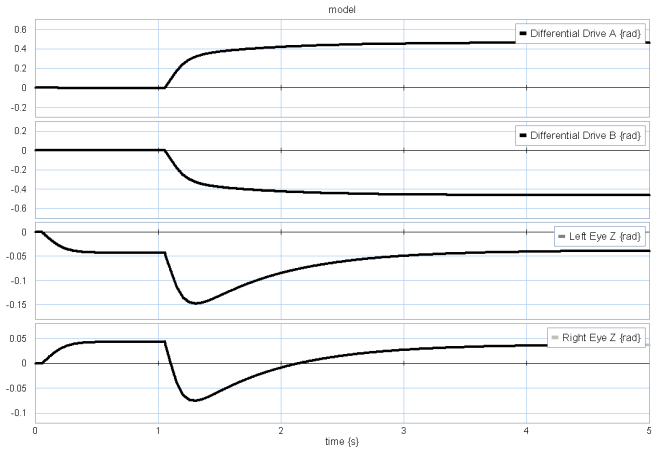

Fig. 11. Time plots of the joint angles for the differential drive and the eyes — The counter rotation of the differential drive actuators (top two) result in a panning motion of the head. The rotation of the eyes (bottom two) with respect to the head is much faster than the head movement and also shows the desired counter rotation to compensate for the head movement once the target is in view.

of the system. The actuators are modeled as closed loop PID-controlled servo motors.

Fig. 11 shows time plots of the joint angles of the differential drive and the z-axis rotation eyes for a horizontal saccade. The norm of the target vector in 2D camera coordinates is reduced to zero in about 400 ms. This is achieved by a rapid movement of the eyes, which reach their maximum angle in about 300 ms. After this, the eyes start to rotate back to their initial position, as the differential drive moves the head towards the target. When this plot is compared with Fig. 2, the resemblance with human behavior is apparent. The plots show similar patterns, but on a different time scale. This is because the rapid eye movement of humans could not be achieved.

The algorithm has been succesfully tested in various situations, e.g.: saccades, target tracking and null-space movement (i.e. moving the head while keeping the target in focus).

VI. Conclusions and Future Work

A motion control algorithm for a humanoid head has been designed. The algorithm acts on camera inputs and can control the humanoid head in a human-like way. This has been achieved by appropriately actuating the redundant joints using a null-space projection method. A dynamic model based on bondgraph and screw theory has been developed and has been used to test the motion control algorithm in a simulated environment. Simulations have shown that both saccades and target tracking tasks can be performed, encountering a human like behavior in several circumstances.

Future work are the implementation of the control algorithms on a real-time hardware platform and evaluation of the performance in a hardware-in-the-loop environment. When the algorithm is found to work correctly on the real-time control platform, the real humanoid head system can be interfaced, because the dynamic model is developed to match the real system as close as possible. No major issues are to be expected in the implementation phase.

References

[1] "Dutch robotics," 3TU Federation, 2008. [Online]. Available: http://www.dutchrobotics.net
[2] J. Bennik, "Mechatronic design of a humanoid head and neck," Master's thesis (unpublished), University of Twente, 2008.
[3] R. Beira, M. Lopes, M. Praca, J. Santos-Victor, A. Bernardino, G. Metta, F. Becchi, and R. Saltaren, "Design of the robot-cub (icub) head," in *Proceedings IEEE International Conference on Robotics and Automation*, 2006.
[4] J. Sabater, N. Garcia, C. Perez, J. Azorin, R. Saltaren, and E. Yime, "Design and analysis of a spherical humanoid neck using screw theory," in *Proceedings IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics*, 2006.
[5] H. Goossens and A. van Opstal, "Human eye-head coordination in two dimensions under different sensorimotor conditions," *Journal of Experimental Brain Research*, vol. 114, no. 3, pp. 542–560, 1997.
[6] S. Stramigioli and H. Bruyninckx, "Geometry and screw theory for robotics," in *Proceedings IEEE International Conference on Robotics and Automation*, 2001.
[7] V. Duindam, "Port-based modeling and control for efficient bipedal walking robots," Ph.D. dissertation, University of Twente, 2006.
[8] R. Reilink, "Saliency based humanoid gaze emulation using a moving camera setup," Master's thesis (unpublished), University of Twente, 2008.
[9] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An invitation to 3-D vision*. Springer, 2006.
[10] K. Doty, C. Melchiorri, and C. Bonivento, "A theory of generalized inverses applied to robotics," *International Journal of Robotic Research*, vol. 12, no. 1, pp. 1–19, 1993.
[11] S. Vijayakumar, J. Conradt, T. Shibata, and S. Schaal, "Overt visual attention for a humanoid robot," in *Proceedings IEEE International Conference on Intelligence in Robotics and Autonomous Systems*, 2001.
[12] "20-sim," Controllab Products B.V., 2008. [Online]. Available: http://www.20sim.com

3 Implementation and Testing

Implementation of a Vision Based Motion Control Algorithm for a Humanoid Head

L.C. Visser, S. Stramigioli, R. Carloni, G. van Oort, E.C. Dertien

***Abstract*—This paper describes the implementation and testing of a vision based motion control algorithm for a humanoid head. The motion control algorithm has been designed in [1] and provides the head a human-like behavior. In this work, we describe the implementation and tests in a simulation environment and the migration of the algorithm from the simulation environment to a real-time control platform. This migration is governed by a step-by-step integration and testing procedure. After each step, the algorithm output is validated and its performance evaluated. The algorithm is implemented succesfully on a real-time PC-104 platform.**

***Index Terms*—Digital control, programming, real time systems, software testing**

I. Introduction

WITHIN the 3TU collaboration project, the Control Engineering group at the University of Twente and research groups of the universities of Delft and Eindhoven, in collaboration with industry partners, are developing the humanoid robot "TUlip" [2]. In the scope of this project, a humanoid head-neck system has been developed, however, the head-neck system will serve two purposes. Firstly, it is meant to be mounted on a teen-sized humanoid robot as a way to percieve the environment. Secondly, the head will be used as research platform in the field of human-machine interaction. The mechanical design consists of a four degree of freedom neck and a vision system based on two cameras. The cameras, representing the eyes, tilt on a common axis and rotate sideways independently.

In previous work a motion control algorithm for this system has been developed and tested on a dynamic model in a simulation environment [1]. This paper will focus on the migration of the motion control algorithm from the simulation environment to an implementation on a real-time platform.

This paper is organized as follows: in Section II a set of requirements for the real-time implementation is presented. Based on these requirements, a number of implementation choices have been made, which are presented in Section III. Section IV presents a step-wise integration and testing method that should lead to a succesful implementation of the motion control algorithm on the real-time platform. The outcome of this method will be presented and discussed in Section V.

II. Requirements

The control of the humanoid head system requires a number of components: a vision processing algorithm, a motion control algorithm and hardware interfaces. The vision processing algorithm will run on a separate, dedicated PC, so implementation of the control algorithm consists of the implementation of the motion control algorithm, the actuator controllers and the design of a framework providing communication between the algorithms and the hardware. The following software components are needed for the realization of a succesful implementation of a controller system for the humanoid head:

- A real-time operating system (RTOS), running on a PC-104 platform that hosts the control software
- A framework running on the RTOS, providing communication between components (i.e. the motion control algorithm, PID-controllers, hardware interface, etc.)
- An implementation of the the motion control algorithm that can be integrated in the framework
- A real-time implementation of the actuator control loops
- An interface to the vision processing algorithm
- An interface to the hardware

These mechanisms and the relations between them is summarized in Fig. 1. The figure also shows a low-level safety layer. In principle, the control algorithms should take care of initializing the system (homing of the encoders, software endstops, etc.) and provide safe operation. Regardless, a back-up, low level safety layer should be present outside the framework. This safety layer may consist of, for example, emergency shutdown buttons, current limiters for the actuators, hardware endstops, etc., so to guarantee safe operation at all time, even when the framework stops functioning (e.g. when the operating system crashes). The implementation of this part is beyond the scope of this paper.

A. Component Requirements

In this paragraph we describe the requirements listed above that are required in order to come to a successful implementation and integration of the motion control algorithm.

Real-Time Operating System: The real-time operating system should be able to provide a hard real-time environment. It should also allow for easy hardware interfacing, since hardware will be used that requires very low-level programming.

Framework: The framework should provide a communication network for the different parts involved in the control structure. Moreover, it should provide an abstract interface to which the modules can be connected, and provide the real-time environment for the modules.

Motion Control Algorithm: The control algorithm will run in a hard real-time environment. As observed in simulations, the algorithm should run at at least 60 Hz., so execution time should not exceed 0.017 s. Since the motion control algorithm is only a small part of the complete system and time is needed

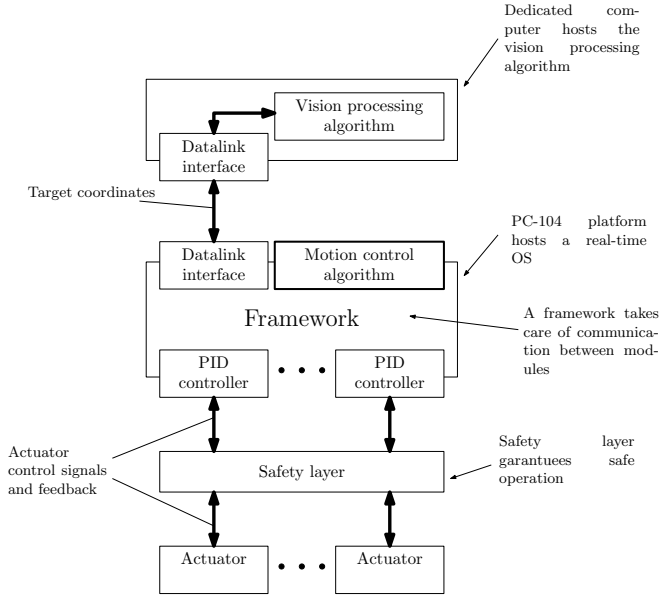

Fig. 1. Overview of the control structure — The framework running on a RTOS provides communication between modules (datalink interface, motion control algorithm, actuator controllers). The framework also serves as an abstraction layer between the RTOS and the modules.

to perform other calculations as well, the maximum available execution time is more likely to be around 0.005 s. This imposes limitations on the programming language used and programming constructions that can be used. For example, interpreted languages or programming languages that run in a managed environment cannot be used.

The algorithm needs to interface the framework to receive input values and output desired joint velocities. This interface needs to be as simple (i.e. little overhead and delay) but robust as possible.

Actuator Control Loops: The actuator control loops consist of an integrator and closed loop PID-control of the actuator position. The integrators generate setpoints for the PID-controllers of the servo actuators. Since there are seven actuators, it is desirable that there is a generic controller that can be instantiated multiple times with different parameters. This is preferable over implementing seven seperate controllers, which is more error-prone. Also, in this way the interface to the controllers is always the same, which simplifies the integration in the framework. The actuator control loops and hardware interfaces should run at a high sample frequency of 1 kHz. or higher to obtain smooth and stable actuation of the system. The integrators in the control loop are essentially interpolating the motion controller output.

Interface to the Hardware: The interfaces to the hardware should be provided in a generic way, in case the hardware drivers change. The hardware interface should wrap around the hardware driver, so changes of the driver do not require changes in the framework.

Interface to the Vision Algorithm: The vision processing will run on a separate PC connected to the PC-104 platform through ethernet. Since the vision processing algorithm will run at a much slower rate than the motion control algorithm ($\sim$ 20 Hz. and $\sim$ 60 Hz. respectively), it is not necessary

Fig. 2. Task execution scheme — There are three real-time tasks that run simultaneously: the motion control algorithm at low frequency (60 Hz. or higher), setpoint generation through integration (1 kHz. or higher) and closed loop actuator control and interfacing (also 1 kHz. or higher). Interfacing the vision algorithm is not a real-time task.

that the interface to the vision algorithm runs hard-real time. It is sufficient for the motion control algorithm to check for new data on the interface once every iteration. This approach requires that the interface is non-blocking, so that the real-time operation of the control algorithm can be guaranteed.

From these requirements it follows that we have three real-time tasks that run simultaneously: a motion control task at at least 60 Hz., an integration task and a closed loop actuator control and interfacing task at much higher frequencies (1 kHz. or higher). The vision algorithm should be interfaced periodically as well, but not necessarily real-time. This results in a task timing scheme as depicted in Fig. 2. The RTOS and framework should provide means to implement this scheme.

III. Implementation Choices

Based on the requirements, a number of choices have been made regarding implementation details.

A. Real-Time Operating System

The choice was made in favor of a GNU/Linux distribution running the Xenomai [3] real-time environment. The primary reason for choosing GNU/Linux is because of the real-time capabilities, the wide range of supported hardware, the ease of hardware interfacing and the flexibility.

B. Framework

The framework is based on the work presented by Lootsma in [4]. He has provided a easy-to-use and flexible interface to the real-time environment provided by Xenomai. It was found that this interface was well-suited for the framework structure envisioned for this project.

The framework is written in C++ programming language. The object-orientated features of this language are very well suited to fulfill the modularity requirement. It also allows for a certain level of abstraction regarding the aforementioned interfaces.

C. Motion Control Algorithm

Based on the requirements, the choice was made to implement the algorithm in the C-programming language. Because C-code should be compiled to native machine code, programs written in C can be very fast and are suitable for hard real-time operations (depending on programming structures used). C-compilers for almost any platform exist, which will make

the algorithm very portable to other platforms. The availability of low-level memory operations allows for a very fast and consistent interface to the framework through pointers. An extra advantage is that the framework is implemented in C++, allowing easy integration of the control algorithm in the framework.

To simplify the implementation of the many matrix operations in the algorithm, an external matrix library is used. The choice was made to use the GNU Scientific Library (GSL) [5]. This is a scientific mathematics library, implemented in C. It is known to compile on many platforms and is actively maintained. It is a mature library that provides a consistent interface and many optimization possibilities. A wrapper interface to this library is made, in case the choice is made to use a different library in the future. In that case, only the interface needs to be changed, while the algorithm can remain unaltered. This should reduce the chance of erroneous operation in case of a change of platform or library.

The motion control algorithm will be embedded in a state machine based controller. In this way, the hardware and software can be properly initialized prior to the execution of the control algorithm.

D. Actuator Control Loops

For implementing the actuator control loops, two options are available: exporting the controller designs from the 20-sim software package [6] or realizing a custom implementation. Although 20-sim provides excellent functionality for C-code generation, the choice was made to manually implement the controllers. The main motivation is that in this way an optimal integration with the framework can be realized.

E. Interface to the Hardware

The hardware is interfaced with a FPGA based PC104+ Anything I/O card [7]. These boards have been used in many projects in the Control Engineering group, so documented drivers and a lot of experience are available.

F. Interface to the Vision Algorithm

The interface to the vision processing algorithm should provide non-blocking read and write functions. Unix-sockets provide this functionality in combination with the `select()` system call [8].

The designed implementation structure and communication signals is shown in Fig. 3. The real-time tasks are grouped by related operations. The interfaces are primairily implemented through memory pointers. In this way, communication overhead is kept to a minimum. Also, certain cases of unintended data manipulation can be discovered at compile time.

Initially, the choice was made to allocate memory for the algorithm dynamically in each call to the algorithm. While this results in overhead (memory allocation is expensive in that it is not deterministic), it was found that in this way the code can be kept organized better and memory usage can be minimized. Tests will need to show whether the overhead caused by the memory allocation operations is too big. In that case, initialization and termination functions should be written that allocate and deallocate appropriate amounts of memory for the algorithm at startup and shutdown.

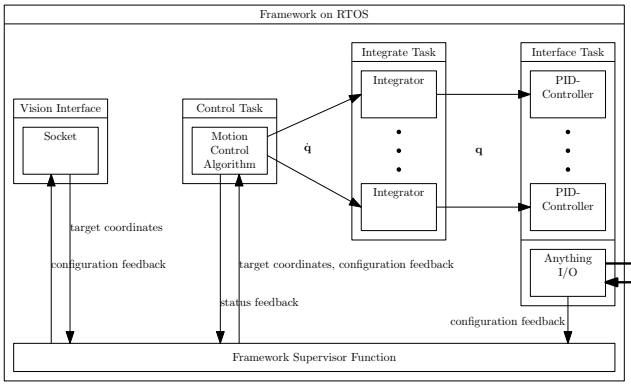

Fig. 3. Overview of the implementation structure — The framework supervisory function controls the initialization phase and provides means of communication between modules. The three tasks that need to run real-time are grouped and will run in separate threads: the motion control algorithm at a low frequency, integration and hardware interfacing at high frequencies.

IV. Stepwise Integration and Testing

In order to come to a correctly functioning implementation of the control algorithm, a stepwise testing and integration strategy has been adopted [9]. This procedure is illustrated in Fig. 4. Point of departure is the control algorithm implemented in the scripted simulation environment of the 20-sim software package. Then the algorithm is ported to C-code and tested again to validate its correctness. After successful completion of this phase, the algorithm can be integrated into the controller framework and should be tested again. The final phase comprises the integration of the seperately designed vision processing algorithm, the motion control algorithm integrated in the framework on the PC-104, and the interface to the hardware.

The framework has been developed and tested separately. The design, implementation and testing procedure for the framework is treated in [4].

A. Testing in a Simulation Environment

The first phase is to exhaustively test the algorithm in a simulation environment. This has been done by implementing the algorithm together with a dynamic model in the 20-sim software package. The algorithm has been tested in a number of predefined scenarios:

- An instanteneous change in target position, resulting in a saccade
- A smoothly moving target that must be tracked
- A randomly but smoothly moving target that must be tracked
- A stationary target that must be kept in view while redundant joints are actuated
- Combinations of the above

During these tests, signals from the (simulated) actuators, PID controllers and cameras were logged, as well as the control

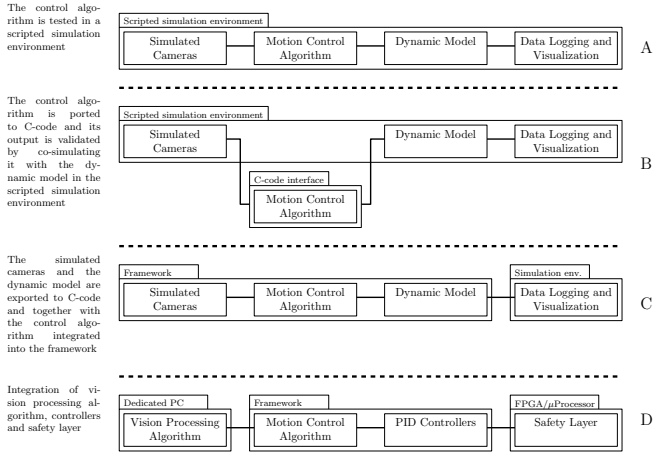

Fig. 4. Stepwise testing and integration procedure — Four separate testing phases are identified, each of which works closer towards integration of the motion control algorithm in the real-time environment of the controller framework.

algorithm output. This data is used in the next phase to validate the correctness of the ported algorithm.

B. Porting and Testing the Control Algorithm

The next phase is to port the motion control algorithm to C-code. The process of porting the algorithm consists of two phases. The first step is to provide an interface to GSL and test this interface. The interface primarily consists of elementary matrix and vector manipulation functions, e.g. addition, multiplication, etc.

One key feature of the algorithm is the utilization of a pseudo-inverse with optimization [10] to obtain desired joint actuator velocities

$$\dot{\mathbf{q}} = \mathbf{F}^{+}\dot{\mathbf{x}} + \left(\mathbf{I}_7 - \mathbf{F}^{+}\mathbf{F}\right)\mathbf{z}. \tag{1}$$

Since GSL does not support pseudo-inverses natively, a routine was implemented that utilizes Singular Value (SV) decomposition to obtain a pseudo-inverse [11]. A matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$ can be decomposed in

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{2}$$

where the diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{m\times m}$ holds the singular values σ_i

$$\begin{aligned} &\mathbf{\Sigma} \triangleq \operatorname{diag}\left(\sigma_1, \ldots, \sigma_p\right) \\ &p = \min\left\{m, n\right\} \\ &\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_k > 0 \\ &\sigma_{k+1} = \cdots = \sigma_p = 0. \end{aligned} \tag{3}$$

From this, a pseudo-inverse $\mathbf{A}^{+} \in \mathbb{R}^{m\times n}$ is obtained through

$$\mathbf{A}^{+} = \mathbf{V}\mathbf{\Sigma}^{+}\mathbf{U}^T, \tag{4}$$

where $\mathbf{\Sigma}^{+}$ follows from (3) and is defined as

$$\mathbf{\Sigma}^{+} \triangleq \operatorname{diag}\underbrace{\left(\frac{1}{\sigma_1}, \ldots, \frac{1}{\sigma_k}, 0, \ldots, 0\right)}_{p=\min\{m,n\}}. \tag{5}$$

GSL only supports SV decomposition for matrices $\mathbf{A} \in \mathbb{R}^{n\times m}$ with $n \geqslant m$ and the matrix $\mathbf{F} \in \mathbb{R}^{4\times 7}$ does not fulfill this condition. To be able to use the GSL decomposition routines, the SV decomposition of $\mathbf{F}^T$ is taken according to (2). Using $\mathbf{\Sigma}^{+}$ from (5), $\mathbf{F}^{+}$ is then obtained from:

$$\begin{aligned} \mathbf{G} \triangleq \mathbf{F}^T &= \mathbf{U}_G\mathbf{\Sigma}_G\mathbf{V}_G^T \\ \mathbf{G}^{+} = \mathbf{V}_G\mathbf{\Sigma}_G^{+}\mathbf{U}_G^T &= \left(\mathbf{F}^{+}\right)^T \\ \Rightarrow \mathbf{F}^{+} &= \left(\mathbf{G}^{+}\right)^T = \mathbf{U}_G\mathbf{\Sigma}_G^{+}\mathbf{V}_G^T \end{aligned} \tag{6}$$

The output of the ported algorithm is validated using the data collected in the experiments conducted in the previous phase. This is done by an automated program that provides the algorithm with input data from simulations of the previous phase. The output of the ported algorithm is compared with the output of the scripted algorithm and an error is raised if the output is invalid.

C. Integration and Testing on the PC-104

The next phase in the integration process is to incorporate the motion control algorithm in the framework on the PC-104. Since both the framework and the control algorithm are written in the C-language, this is a fairly easy task. In order to test the algorithm on the framework, it needs to receive target coordinates and joint actuator position feedback. This is achieved by porting the dynamic model and simulated target and cameras to C-code as well. The 20-sim software package provides template-based export functions to do this automatically. It should be noted that, due to the model complexity, only a limited number of scenarios that do not demand to much of the model can be tested and only in simulated real-time. This is because the PC-104 platform has limited computing power available. Nonetheless, this should be sufficient to verify correct behavior of the control algorithm.

In this phase, where both the control algorithm and the dynamic model run on the PC-104 platform, a PC is used to gather and visualize data signals. Since the correctness of the algorithm has alread been validated in the previous phase, the focus is primarily on testing the interface between the framework and the algorithm.

D. Connecting the Vision Processing Algorithm

The final phase of the testing and integration process is to connect the seperately developed vision processing algorithm to the framework and control algorithm. The vision processing algorithm analyzes image frames from two cameras and decides what the head should look at. Due to the nature of this task, this phase requires that the dynamic model is removed from the framework and that the actual hardware is interfaced.

If all testing phases have been completed succesfully, this phase will yield the complete integrated system, comprising cameras, vision processing algorithm, motion control algorithm and a naturally moving humanoid head.

V. Results and Conclusions

The first three steps of the integration and testing procedure have been completed. The algorithm was successfully ported to C-code and the validation process showed no errors in the calculated output. The execution time of the algorithm was found to be approximately 40 μs, significantly below the maximum of 5 ms. As mentioned before, there is still room for improvement if 40 μs turns out to be too long. Also, since memory allocation is not deterministic, the required execution time of the algorithm cannot be determined beforehand with the current implementation. This might need to be improved in the future as well.

Integration of the motion control algorithm with the controller framework has been completed succesfully as well. Because the PC-104 platform has a limited amount of calculation power, it was not possible to simulate the algorithm and the dynamic model real-time. Instead, a pseudo real-time environment, where the clock runs a fixed factor slower to "create" time, was created in which the algorithm was tested. This only allowed for very simple testing scenarios, but the emphasis in this phase was on testing the integration with the framework rather than on the correct behavior of the algorithm (since this was already validated). The scenarios that have been tested have shown no problems.

The final integration step, where the motion control algorithm and vision processing algorithm are connected and the real hardware is interfaced has not been completed yet. This is due to unavailability of the hardware. No serious problems are expected to arise in completing the integration, since extensive testing of the algorithm and the integration in the framework have not revealed any issues. It is therefore expected that the final integration step will be succesfull as soon as all parts are available.

References

[1] L. Visser, S. Stramigioli, G. van Oort, and E. Dertien, "Design of a vision based motion control algorithm for a humanoid head," Master's thesis (unpublished), University of Twente, 2008.
[2] "Dutch robotics," 3TU Federation, 2008. [Online]. Available: http://www.dutchrobotics.net
[3] P. Gerum, "Xenomai - implementing a rtos emulation framework on gnu/linux," 2004,2008. [Online]. Available: http://www.xenomai.org
[4] M. Lootsma, "Design of the global software structure and controller framework for the 3tu soccer robot," Master's thesis (unpublished), University of Twente, 2008.
[5] M. Galassi, "Gnu scientific library reference manual (2nd ed.)." [Online]. Available: http://www.gnu.org/software/gsl/
[6] "20-sim," Controllab Products B.V., 2008. [Online]. Available: http://www.20sim.com
[7] "4i65/4i68 fpga based pc104-plus anything i/o card," Mesa Electronics, 2008. [Online]. Available: http://www.mesanet.com/
[8] *Linux Programmer's Manual (2): SELECT*, 2008. [Online]. Available: http://www.linuxmanpages.com/man2/select.2.php
[9] J. Cooling, *Software engineering for real-time systems*. Addison-Wesley, 2003.
[10] A. Liegeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 7, no. 12, pp. 868–871, 1977.
[11] Y. Nakamura, *Advanced robotics: redundancy and optimization*. Addison-Wesley, 1991.

A Screw Theory

This appendix summarizes some basic concepts of screw theory in order to allow the reader to better understand and follow the formulae presented in this report. For a more comprehensive and complete introduction, the reader is referred to literature.

A.1 Homogeneous Coordinates

Let

$$\mathbf{p}^i = \begin{bmatrix} x \\ y \\ z \end{bmatrix}^i$$

be a generic point in three dimensional Euclidian space $\mathcal{E}(3)$, expressed in coordinate frame Ψ_i. The point can be numerically expressed in a different coordinate frame Ψ_j by applying a change of coordinates $\mathcal{E}(3) \rightarrow \mathcal{E}(3)$ due to a rotation and translation

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^j = \mathbf{R}_i^j \begin{bmatrix} x \\ y \\ z \end{bmatrix}^i + \mathbf{d}_i^j,$$

where $\mathbf{R}_i^j \in SO(3)$ defines the rotation from Ψ_i to Ψ_j, and the vector $\mathbf{d}_i^j \in \mathbb{R}^3$ the corresponding translation.

The change of coordinates can be expressed by a single matrix multiplication by introducing homogeneous coordinates. A generic point $\mathbf{P}$ in homogeneous coordinates is represented by

$$\mathbf{p}^i = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^i .$$

The change of coordinates is then defined by

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^j = \mathbf{H}_i^j \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^i ,$$

where the homogeneous matrix $\mathbf{H}_i^j \in SE(3)$ has the following structure:

$$\mathbf{H}_i^j = \begin{bmatrix} \mathbf{R}_i^j & \mathbf{d}_i^j \\ 0\,0\,0 & 1 \end{bmatrix}.$$

A.2 Twists and Wrenches

It can be shown that a change of coordinates $\mathbf{H} \in SE(3)$ can be found by taking the exponential of a matrix $\tilde{\mathbf{T}} \in \mathfrak{se}(3)$:

$$\mathbf{H} = e^{\tilde{\mathbf{T}}},$$

where $\tilde{\mathbf{T}}$ has the form

$$\tilde{\mathbf{T}} = \begin{bmatrix} \tilde{\omega} & \mathbf{v} \\ 0\,0\,0 & 1 \end{bmatrix},$$

with $\tilde{\omega} \in \mathfrak{so}(3)$ and $\mathbf{v} \in \mathbb{R}^3$. The equivalent vector representation of $\tilde{\mathbf{T}}$ is

$$\mathbf{T} = \begin{bmatrix} \omega \\ \mathbf{v} \end{bmatrix}.$$

Here ω represents a rotational velocity and $\mathbf{v}$ represents a linear velocity.

The vector $\mathbf{T}$ is called twist and represents the generalized velocity of a coordinate system rigidly attached to a body. According to Chasles' theorem, any twist can be written as

$$\begin{bmatrix} \omega \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \omega \\ \mathbf{r} \wedge \omega \end{bmatrix} + \lambda \begin{bmatrix} \mathbf{0} \\ \omega \end{bmatrix}.$$

This corresponds geometrically to a rotating motion around ω combined with a simultaneous translation λ along ω: a screw-like motion (hence the name "screw theory").

If a matrix $\mathbf{H}_i^j \in SE(3)$ defines the change of coordinates from Ψ_i to Ψ_j, then their relative motion, expressed in Ψ_i, is given by:

$$\mathbf{T}_i^{j,j} = \dot{\mathbf{H}}_i^j \mathbf{H}_j^i.$$

It can be shown that $\mathbf{T}_i^{j,j}$ can be expressed in an arbitrary coordinate system Ψ_k:

$$\mathbf{T}_i^{k,j} = \begin{bmatrix} \omega_i^{k,j} \\ \mathbf{v}_i^{k,j} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_j^k & \mathbf{0} \\ \tilde{\mathbf{d}}_j^k \mathbf{R}_j^k & \mathbf{R}_j^k \end{bmatrix} \begin{bmatrix} \omega_i^{j,j} \\ \mathbf{v}_i^{j,j} \end{bmatrix}.$$

This change of coordinates is defined by an operator, called adjoint, on the homogeneous matrix $\mathbf{H}_j^k$ that defines the change of coordinates from Ψ_j to Ψ_k:

$$\mathrm{Ad}_{\mathbf{H}_j^k} \triangleq \begin{bmatrix} \mathbf{R}_j^k & \mathbf{0} \\ \tilde{\mathbf{d}}_j^k \mathbf{R}_j^k & \mathbf{R}_j^k \end{bmatrix}.$$

Similarly, a wrench represents the generalized force acting on a body and takes the form

$$\mathbf{W} = \begin{bmatrix} \mathbf{m} & \mathbf{f} \end{bmatrix}.$$

Here $\mathbf{m}$ represents a torque and $\mathbf{v}$ represents a linear force. A wrench $\mathbf{W}$ is a element of the dual space $\mathfrak{se}(3)^*$ and we have

$$\mathbf{P} = \mathbf{W}\mathbf{T},$$

where $\mathbf{P}$ denotes power.

B Motion Control State Machine

The motion control algorithm is governed by a state machine. The primary task of the state machine is to ensure that the encoders are properly initialized prior to operation. The structure and functional design of the state machine is shown in figure 1. The following states are available:

Stopped This is the initial state in which the state machine will be on startup. Zero output is sent to all encoders. This state can only be left by the start and initialize commands.

Initialize This is a transition state provided for optional logging purposes. It is directly left to go into the homing state of the first joint.

Homing Joint $< n >$ In this state, the endstop detection is used to termine the range of the joints. This is achieved by actuating the joint slowly in positive and subsequently in negative direction until the endstops are hit. By reading the encoder data, the range of the joint can be determined, upon which the joint states can be properly initialized.

Initialization done This state is provided as a pausing point, so that all initialization functions can be completed. Upon receiving a start or stop command, this state will be left to enter the stopped or running state respectively.

Running In this state, the motion control algorithm will calculate actuator velocities as it is designed to do. On any error, this state will be left for the error state. Upon receiving the stop command, the state will be left for the stopped state.

Error This state is provided for error handling. Subsequently the stopped state will be entered.

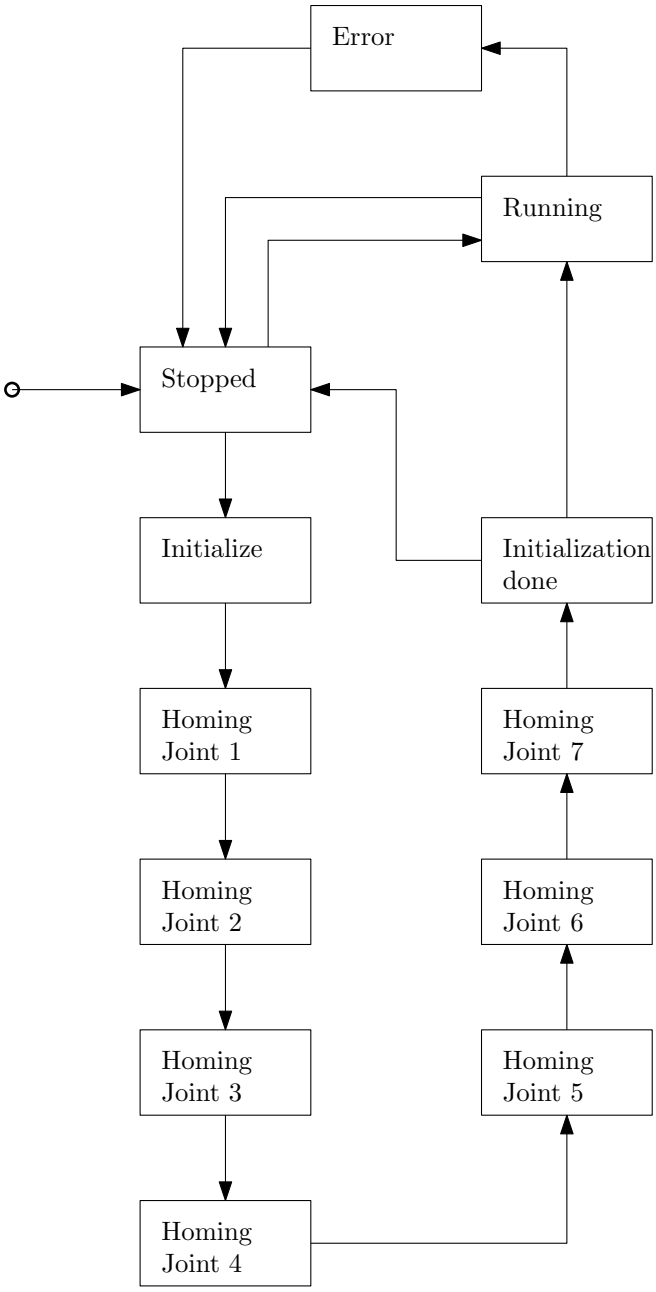

Figure 1: State machine structure.