Shared Control of Mechanical Systems in Virtual Environments

Anders Hansson

November 25, 2009
Master's Thesis in Computing Science, 30 credits
Supervisor at CS-UmU: Thomas Hellström
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

In this master thesis a framework for control of mechanical systems is developed. The framework is extended with possibilities to simulate mechanical systems in a virtual environment. The virtual environment provides an easy way to experiment with high level control algorithms for mechanical systems and human machine interfaces.

This work give a proposal on how the working cycle for an operator of a forwarder crane can be simplified with automated tasks and how shared control between the operator and the automated tasks can be implemented. To evaluate and test the automated tasks together with the shared control implementation a *Valmet 830* forwarder is modeled in the virtual environment.

The developed framework is an object oriented API (Application Programming Interface) written in `C++` called `atvs`. The virtual environment extension is a dynamic real time simulation based on the AgX physics engine [1] and the OpenSceneGraph 3D graphics toolkit [2].

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

There is a huge development of autonomous and semi-autonomous machines for the industry. The mining industry already have autonomous loading and transportation of resources that are supervised from a control room. There are also autonomous vehicle applications for agriculture and personal cars [8].

Most of the autonomous machines operates in known environments and take advantage of that. The autonomous machines that operate in the mine already know the structure of the mine and what it can expect to run into. For machines that operates in unknown environments the autonomous tasks becomes a great challenge. If the environment also is rough and complex the tasks becomes even more complicated. Completely autonomous machines in these environments is very difficult to accomplish. Instead semi-autonomous or manually operated machines are used in these environments. A semi-autonomous machine have some tasks in its working cycle that are automated, but still needs to be supervised by an operator.

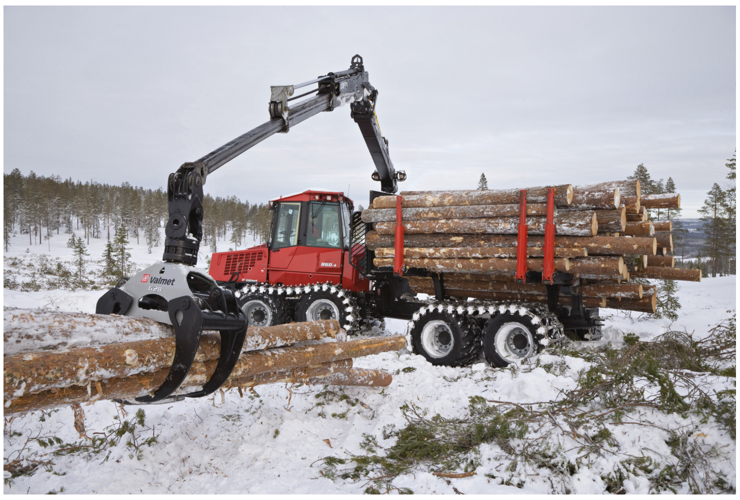


Figure 1.1: Valmet 860.4 forwarder.

The forest environment is an example of an unknown, complex and rough environment. Operators of forestry machines are faced with many fast decisions during a working cycle. Due to the environmental issues and all the fast decisions, the forest machines are almost completely manually controlled. Which, together with the fact that the industry of forest machines are quite young, makes for huge development capabilities in the autonomous field. Today the harvester have felling and logging tasks automated [9] while the forwarder still is completely manually controlled. The automation of tasks in a forest machines work cycle can decrease learning time, increase productivity, prevent physical and mental strain [10], [11]. At Umeå university there are research on automation of forwarder and harvester cranes [12] and autonomous navigation for vehicles in forest environments [8].

During an automated task the operator may want to adjust or control the automated task without interrupt it. When an automatic control system work at the same time as a human operator it is referred to as shared control. They do not necessary work at the same task, but in the most common cases they do.

If shared control should be possible there must be smooth transitions between manual and automatic control. This require that information about intentions is transferred in both ways between operator and the automatic control system. Transferring intentions from operator to automatic control system can be done with the same control interface as the operator use in manual control, which for forest machines mainly are two joysticks. But how should intentions from automatic control system to the operator be transferred? This issue is addressed in [9] where force feedback in the operators joystick is proposed as a solution. Intentions from the automatic control system can also be transferred by graphical indications and sound. Graphical indications can be visualized on a computer display or a HUD (Head Up Display) [9].

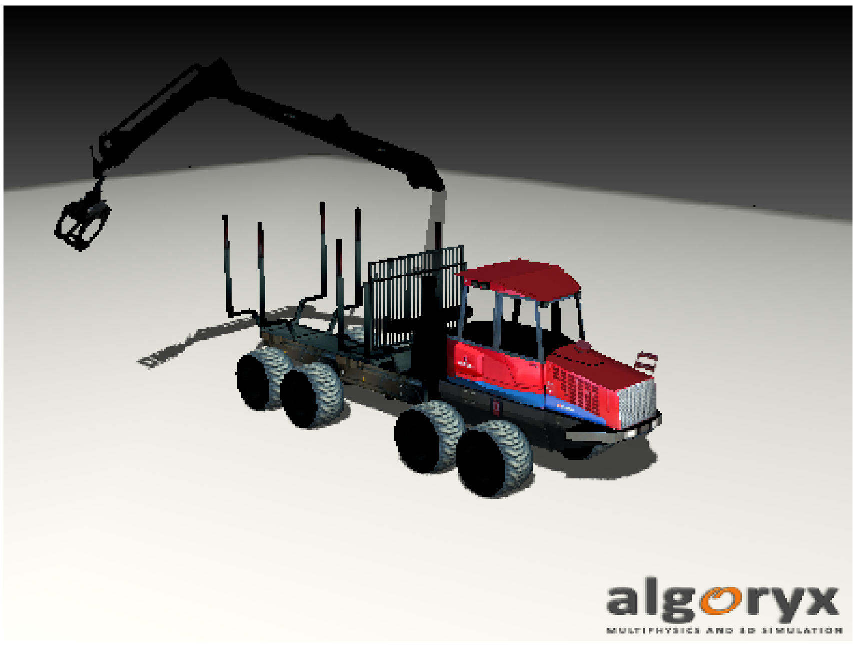


Figure 1.2: Valmet 830 forwarder in the virtual environment.

In this master thesis a simulation framework for mechanical systems called `atvs` is developed. The purpose of `atvs` is to provide an easy way to test and experiment with high level control algorithms and human machine interface, especially in the shared control field. The framework have support for ordinary gaming joysticks and force feedback devices used in ordinary computer games.

A virtual environment for simulation of mechanical systems is connected to the simulation environment. The virtual environment API is called `atvsVE` and is implemented with AgX multiphysics toolkit [1] and OpenSceneGraph [2]. Components for automation of mechanical systems in a virtual environment have been developed around components from the AgX multiphysics toolkit. The implemented components are mainly sensors, actuators and PID controller.

This thesis is concentrated upon the forestry, especially hydraulically actuated cranes on forest machines. Two mechanical systems are implemented for experiment with shared control. A loading system that operates in two dimensions and is supplied with a rotatable grapple. The loading system is used to load or unload logs from a vehicle. Figure 7.1 shows the loading system. The other system is a complete Valmet 830 forwarder with hydraulically actuated crane. A forwarder is a vehicle that collect felled logs in a forest environment and transport them to a road where the logs are unloaded to wait for a lumber car. Figure 1.1 shows a real forwarder and figure 1.2 shows a forwarder in the virtual environment.

1.1 Participants

Short description of companies and organizations related to this thesis.

IFOR [13] With IFOR as a base, Umeå University, Skogforsk and SLU does research and development in collaboration with the industry. IFOR is acronym for Intelligent Off-Road Vehicles (in Swedish: Intelligenta Fordon Off-Road).

Umeå University [14] At Umeå University there is research in robotics, control system and visual interactive simulation that is related to IFOR.

Oryx Simulations [4] A company that produce training simulators for forest machines and other heavy machines. This thesis make use of 3D-models used in Oryx simulators.

Algoryx Simulations [1] Develop the multiphysics engine AgX. The virtual environment in this thesis is based on the AgX engine.

Komatsu Forest [15] The machine manufacturer that develop Valmet forest machines. Komatsu Forest have supplied with drawings and data for the forest machines.

1.2 Outline

Chapter 2 introduce virtual environments and the different control architectures from manual control to semi-autonomous control. In chapter 3 a detailed description of the problem is given. Section 3.4 briefly summarize previous work in the shared control area.

Chapter 4 contains relevant theory for this thesis. Simulation of multibody dynamics is covered in section 4.1. Section 4.2 deals with basic control theory, mainly PID control. In section 4.3 the forward kinematics and inverse kinematics problems of a redundant manipulator are considered. If the reader is familiar with the contents in the sections of chapter 4, they may be skipped.

In chapter 5 implementations of the software components are described. Chapter 6 gives proposals on how shared control can be implemented. Results are presented in chapter 7.

Chapter 2

Background knowledge

This chapter introduce the reader to virtual environments and control architectures.

2.1 Virtual environments

The terms virtual environment and virtual reality are often used interchangeably with the same meaning. The terms refer to a computer generated 3D world in which the users can act, interact and perceive themselves to be in. Often a virtual environment tries to simulate the reality, but there are also applications where that is not the case. Computer games exemplifies both these categories, there are flight games, car racing games, first person shooting games etc, that more or less tries to simulate the reality.

In the huge online multi player role-playing game *"World of warcraft"* [3] the players is part of a world wide virtual society where they can interact with other players and cooperate to carry out missions. A world in *"World of warcraft"* have large extension in space and includes thousands of players that act in the world. It is based on the fantasy Warcraft universe from the predecessor *"Warcraft"* strategy games.

Figure 2.1: Screen shots from the game *"World of warcraft"* [3].

Games often have very nice visual appearance that can be very near to reality, but they usually lack in the dynamic simulation. To make the game playable with a good feeling it may be necessary to make limitations in the dynamic simulation and allow cases that is not allowed in the real world. With realistic dynamic simulations some games tends to feel slow and can be difficult to control. This problem can be exemplified with a car racing game with realistic dynamic simulation. The car would probably be very difficult to control for a person playing the game at the first time and it would take very long time learn to control the car. To make the game more responsively with easily control of the car, modifications are done to the dynamic simulation that not accord with the laws of physics. Computer games are most commonly run on a standard PC and are visually experienced on a computer display. This type of virtual reality is called desktop VR.

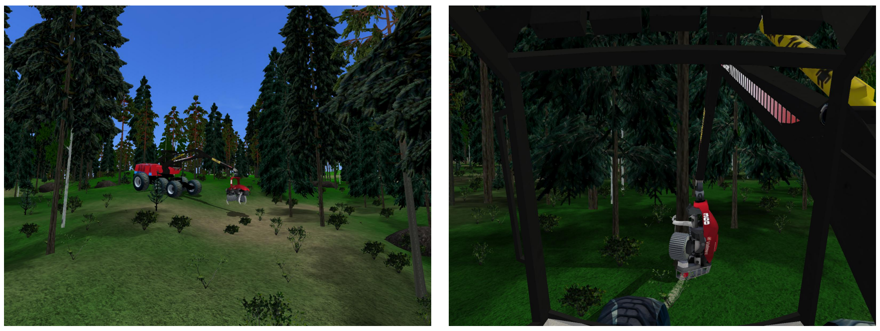
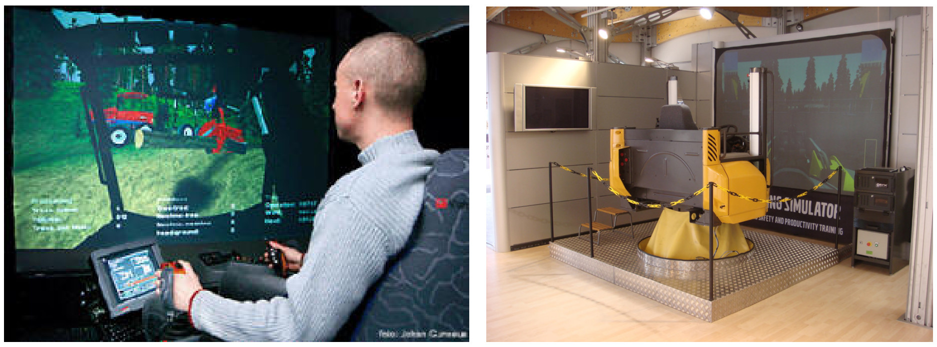

(a) Valmet 911 harvester in terrain. (b) Valmet 911 from inside.

Figure 2.2: Oryx Simulations training simulator for the Valmet 911 harvester [4].

Another application of virtual environments are training simulators. Training simulators have on the other hand very high demands on both visual appearance and the dynamic simulation. Pilots, operators of heavy vehicles and even surgeons can all be trained in simulators before the actual task is performed in the real world. Figure 2.2 shows a training simulator for a Valmet harvester from Oryx Simulations [4]. A simulation of the reality usually gets more realistic and increase the users presence the more senses that can receive inputs from the virtual environment. The visual appearance is very important for a virtual environment, if it is then completed with audio, haptics and something that triggers the sense of balance the user can actually believe it is inside the



(a) (b)

Figure 2.3: Operator environment in Valmet harvester simulator and motion platform for the Volvo wheel loader simulator, both from Oryx simulations [4].

virtual environment. This phenomena is usually called virtual presence [16].

For vehicle training simulators the user's experience and presence increase if the user can act and interact in the virtual environment from an environment similar to the environment in the vehicle. Figure 2.3a shows the operator environment for the harvester simulator, which is identical to the cabin environment in a real harvester. In figure 2.3b the operator environment for an Oryx Volvo wheel loader simulator is shown. The wheel loader cabin is mounted on a motion platform, which makes it possible to move the cabin according to the vehicle cabin in the virtual environment. The motion platform is hydraulically actuated and have 6 degrees of freedom, which makes it possible to create realistic movements of the cabin.

To further increase the users presence the information of motions from their head, arms and legs can be tracked and used in the virtual environment. The motion of the head can be used to manipulate the cameras view point. Motion of the arms can be used to position virtual arms etc. The user can also be supplied with haptic equipment that makes it possible for the user to feel objects in the virtual environment. Instead of using a computer display as the visual medium a head mounted display can be used. The head mounted display completely surrounds the user and makes it visually unaware of the real world.

If some of the above mentioned techniques to increase the users presence does not behave as expected by the user, for example the motion of the camera does not completely coincide with the motion of the head or the motion of the motion platform does not coincide with the motion of the cabin in the virtual environment, the user can feel unpleasant and even wambling. Head mounted displays can also give rise to this unpleasant feeling, which is called simulator sickness and is very important to avoid.

Other areas where virtual environments are used are CAD, architectural models, urban planning, movies etc.

2.2 Control architectures

Systems that includes a human that control a robot device is called *master-slave systems*, where the operator and the instruments used to control the robot device is the *master* side and the controlled robot device is the *slave* side. Between the *master* and *slave* sides there is a communication interface which makes it possible to separate the *master* from the *slave*. If the *master* and *slave* are at different locations it is referred to as *teleoperation*. If the *slave* can transmit forces as feedback to the *masters* controlling device the *slave* is said to be bilaterally controlled. The *master* and *slave* sides can also be at the same location, they can even be at the same device. Machines with operator controlled manipulators, such as wheel loaders, excavators and forest machines, can be *master-slave systems* where both sides resides at the same machine.

In figure 2.4 a teleoperation system form the *master-slave system* where the operator is at *master* side and the robot device are at the *slave* side. The interface between them is some kind of information media that allows information to be transferred between them.

The control architectures *direct control*, *shared control* and *supervisory control* are related to responsibility of control and level of control for the *master* and *slave* as shown in figure 2.5. The figure shows a teleoperation scenario where the human operator is the *master* and the teleoperator is the *slave* [5].

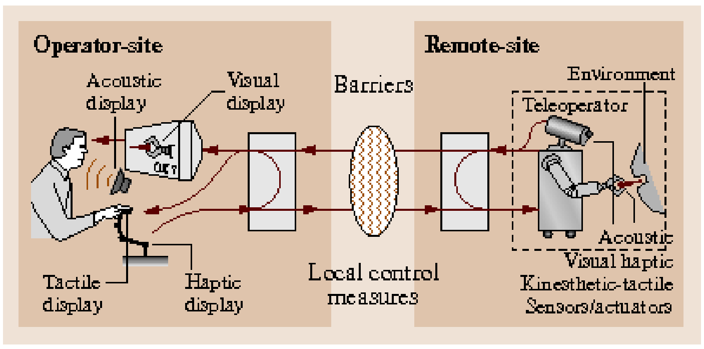


Figure 2.4: A teleoperation system where the operator site is the master and the remote site is the slave. Figure from [5]

2.2.1 Direct control

Direct control does not have any local control loop on the *slave* side. The operator on the *master* side directly controls the individual parameters on the *slave* side. For example the operator controls each joint in a manipulator separately.

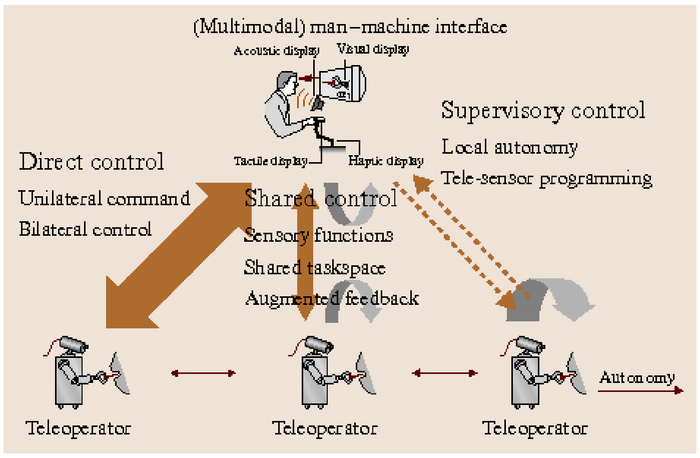


Figure 2.5: Control architectures. Figure from [5]

2.2.2 Shared control

The shared control architecture allows an operator on the *master* side and an autonomous control loop on the *slave* side to control the same task. With a shared task space the operator can control the *slave* with high-level commands to autonomously execute tasks but still have the possibility to affect the tasks at the same time as the *slave* operates on the task. This is done without interrupting the *slave*.

It is also possible for the *slave* to perform calculations based on sensor inputs in the control loop and inputs from the operator to generate control signals. This method can be used to hide complicated control algorithms in the *slaves* control loop and provide a simple interface to the *master*. When the slave is responsible for force control while the operator is responsible for control of the motions is one example. Another example is velocity/position end effector control of redundant manipulators [10, 17], where the operator gives commands in the Cartesian coordinate space and the *slave* transforms it to control signals in the joint space. Shared control can consist of varying degrees of combinations of both direct control and supervisory control. Shared control is illustrated in the middle of figure 2.5.

2.2.3 Supervisory control

A semi-autonomous machine, which have tasks automated but is controlled by an operator goes under this control architecture. The supervisory control architecture has a local autonomous control loop on the *slave* side that is controlled with high-level commands from the *master* side. The human operator on the *master* side is provided with state and model information from the *slave*, from this information the operator supervise the autonomous *slave*. This is illustrated to the right in figure 2.5.

Chapter 3

Problem description

As established in chapter 1 it is difficult to create autonomous machines that operates in unknown and rough environments with a lot of fast decisions included in the working cycle. Therefore direct controlled or semi-autonomous machines are used instead. A semi-autonomous machine have some of its tasks automated. During the execution of an autonomous task it can be necessary for the operator to interact with the task. This is referred to as shared control. This master thesis should develop virtual environment software tools for experiments on machines with semi-autonomous tasks and shared control.

3.1 Purpose

The purpose of this project is to

– Provide tools for experimentation in virtual environments for interfaces between a human operator and a machine with semi-autonomous control.

– Give proposal on how shared control can be realized on a forest machine supplied with a semi-autonomous hydraulically actuated crane.

– Increase the understanding of the transfer (both ways) of control and automation strategies and system models between virtual environments, engineering and real machines.

3.2 Objectives

This project should result in the following

– Algorithms and software tools for experimentation of shared control of terrain vehicles in virtual environments based on the AgX physics engine.

– Demonstration of shared control with force-feedback joysticks. Proposal for further studies in simulator and field.

– Test and development of computational methods for inverse kinematics and shared control suitable for virtual environments.

Assume any high precision sensor- and vision data are available, the above objectives arise some questions. What kind of tasks should be automated and how should this be done? How can a machine and a human operator share control over the vehicle? What methods for shared control are there? How can transition of the control between the human operator and the machine be done in a smooth way? How can the system aid the operator in controlling the machine? How should the operators input signals be combined with the machines input signals?

With the simulation environment resulting from this project, various approaches for semi-autonomous machines with shared control and force feedback can be evaluated to find answers to the above questions.

3.3 Components

Mechanical systems in the virtual environment should be simulated with AgX physics engine and be modeled with rigid bodies and constraints. Figure 3.1 shows how the Valmet forwarder in this thesis is modeled. Actuators of hinge- and linear motor type should be implemented for the virtual environment, as well as PID control. The term actuator will be used interchangeably with the term motor throughout this text.

Rigid bodies and collision geometries

A rigid body is a representation of a finite solid body that can not be deformed. The rigid body have a position in space, translational- and rotational velocities and mass properties. Its physical behavior is determined from the mass properties.
To be able to perform collision detection one or several collision geometries must be attached to the rigid body. A collision geometry is a shape that defines the body's extension in space. In figure 3.1 collision geometries appears as green shapes while the center of mass for the rigid body appears as a small blue sphere.

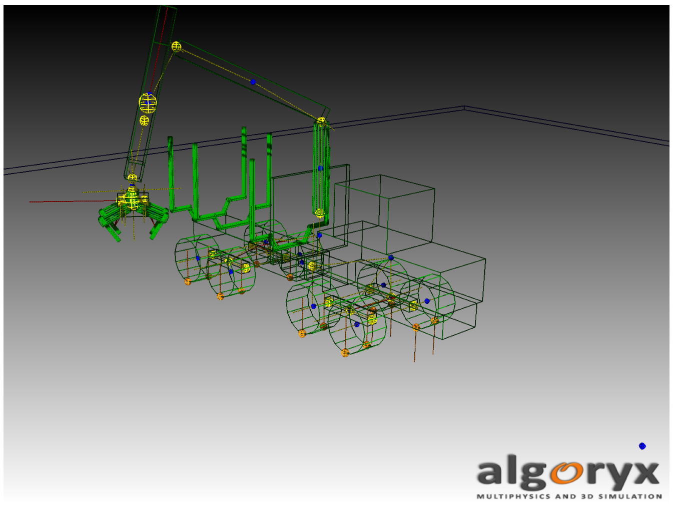


Figure 3.1: Valmet 830 forwarder modeled with rigid bodies and constraints.

The AgX physics engine allows several collision geometries to be associated with a rigid body, the rigid body's mass properties are then determined from the size and form of the collision geometries.

Constraints

A constraint remove degrees of freedom for a rigid body. Constraints can be used to connect two rigid bodies by remove degrees of freedom between them. Two important constraints are hinge- and translational constraints, also called revolute- and prismatic constraints. The hinge constraint let two bodies rotate relative to each other around an axis. The translational constraint let two bodies move relative to each other along an axis. Hinge and prismatic constraints as well as several other types of constraints are included the AgX toolkit. In figure 3.1 constraints appears as yellow spheres, the constraint axis is shown with a yellow line.

Actuators, Sensors and PID control

It should be possible for simulated actuators to be connected to the hinge and translational constraints. The actuators should be simulated with a motor function that applies torques or forces to participating bodies.

For the hinge and prismatic constraint a position and speed sensor should be implemented. For hinge constraints the position sensor should measure the angle in radians and the speed sensor should measure the the magnitude of the rotational velocity in the hinge. For the translational constraint the position sensor measure the displacement between the two participating bodies along the constraint axis and the speed sensor measure the magnitude of the displacement velocity. To be able to control the mechanical system a PID control algorithm should be implemented.

Note that it is possible to let the constraint solver in AgX control velocity and angle in a constraint, but this should not be used. Instead a more realistic behavior, where the actuator is controlled with a control signal to generate forces or torques is desired.

Inverse kinematic algorithm

To simplify high level control commands for a robot arm an inverse kinematic algorithm must be implemented. Inverse kinematic makes it possible to calculate joint velocities for a given end effector velocity or joint positions for a given end effector position. This should be done for a forwarder crane, which is a redundant manipulator. The forwarder crane is redundant because it have four joints, excluded the grapple with rotator, but only operates in three dimensions. Consequently it have one joint more than needed and there typically exists infinitely many joint configurations for one position of the crane tip.

3.4 Related work

The teleoperation field has a huge part in the development of the control architectures described in section 2.2. Underwater applications that force the operator to control the applications from above the water surface, space applications where the operator resides in a control room back on earth are examples of teleoperation systems that have contributed to the development of the control architectures. Also teleoperation applications for high radioactive areas have contributed to the development. Delays between the operator and the application, as in space applications, have favored the development of semi-autonomous systems and systems with shared control and force feedback. In [5] an historical review of teleoperation and applications that have favored the development in the field are given.

In [18] a bilateral shard control architecture is developed to control a machine. An experiment setup where a virtual car is controlled with a haptic steering wheel, which aids the operator in path following tasks, is used to evaluate the architecture. Results from the experiment shows that the haptic steering wheel decrease the demands on visual feedback, increase the path following performance and improves the reaction time.

End effector velocity control with stiffness force feedback calculated at the end effector for an excavator is examined in [19].

In [20, 21] a commercial wheelchair, the NavChair, is equipped with a shared control navigation system where the human and machine shares the control over the wheelchair in the way that the wheelchair can automatically adapt to human behaviors. The NavChair is built on an obstacle avoidance algorithm specifically developed to meet the systems shared control needs.

In [22] the force control of a contour following manipulator is automatically controlled by the *slave* while velocity and direction is controlled by a human operator.

A very similar rigid body dynamic simulation framework is developed and used in [6]. It is used mainly for robot motion planning in rough terrain and also have an interface to simulink [23]. It is a MathWorks software for modeling and simulating dynamic systems based on the Matlab environment.

Chapter 4

Related theory

This chapter introduce theory that are related to this thesis. In section 4.1 dynamic simulations with multiple rigid bodies and constraints are briefly described. Basic theory for control systems and PID control are introduced in section 4.2.
Section 4.3 discuss forward and inverse kinematics. If these topics already are familiar to the reader, the reader can continue to chapter 5.

4.1 Simulation of multibody dynamics

This section briefly describes the basics of simulating multibody dynamics, see [24] for a more detailed study. The term *multibody dynamics* refer to the physics of multiple rigid- or deformable bodies that may be connected to each other. Here only rigid bodies are considered.

To simulate multibody dynamics, ordinary differential equations (ODE) together with algebraic constraints forms differential algebraic equations (DAE) describing motion of bodies. These have to be time–discretized and solved, which is referred to as time integration. The motion of the bodies depends on external forces and constraint forces that emerges from contacts and joints.

Collision detection determine contact information between penetrating bodies, which then is used for solving collisions and contacts. Solving for collisions and contacts as well as constraint forces exerted from the joints typically involves to solve a large sparse matrix system. There are specialized software libraries available to solve these sparse systems, one of those are *UMFPACK* [25].

A system of linear equations can be solved with direct or iterative methods. If a solution exists a direct method finds the solution in a finite number of operations. Iterative methods improves the solution each iteration, but they typically converge very slowly. Therefore iterative methods often are rejected. But an iterative method typically produce an enough accurate solution very fast, which makes it to a pleasant choice for simulation of multibody dynamics. Iterative methods are also efficient in solving large sparse matrix systems.

4.1.1 Simulator paradigms

Three commonly occurring simulator paradigms are Penalty-based methods, constraint-based methods, impulse-based methods. Then there also exists hybrids among simulator paradigms.

Penalty-based methods

Penalty-based methods models physical behavior with springs and dampers. For example when rigid bodies penetrate each other, the penetration is penalized with a force computed by a spring and damper system. The penalty-based method can also easily be extended to handle soft bodies.

In contrast to the constraint-based and impulse-based methods the penalty-based methods allows for penetration of bodies. Deep penetrations may cause stiff ODEs, which means that a numerical method for solving the equation can be unstable, except for very small time steps. Therefore penalty-based methods require small time steps to avoid numerical instability and to keep penetrations minimal [24, 26].

Impulse-based methods

An impulse based method model the physics with collision impulses and do not allow for penetrations. A static contact is modeled with very small high frequent occurring collision impulses. Impulses is applied at the closest points between two colliding objects, and therefore contact determination is not needed. The method is computationally effective except for static contacts and is often a good choice for real time simulators [24, 26].

Constraint-based methods

The constraint-based methods does not allow for penetration and handles static and continues contacts very well. Contact forces are generated with non-holonomic unilateral constraints

$$\Psi(\mathbf{s}) \geq 0 \tag{4.1}$$

where $\mathbf{s}$ is the generalized position vector described in 4.1.4. As with holonomic constraints, see section 4.1.4 for a more detailed description of the following, a kinematic constraint is achieved by differentiate with respect to time

$$\mathbf{J}_\Psi\, u \geq 0, \tag{4.2}$$

where $\mathbf{J}_\Psi$ is the Jacobian matrix. The constraint forces are then

$$\mathbf{f}_{\boldsymbol{\Psi}} = \mathbf{J}_\Psi^T\, \boldsymbol{\lambda}_\Psi, \tag{4.3}$$

in the same way as for holonomic constraints, where $\boldsymbol{\lambda}_\Psi$ is a vector of Lagrange multipliers and the superscript ^{T} denotes the transpose. When solve for collisions an additional impulse is included besides the constraint.

Hybrids

The benefits of several methods can be combined into hybrids of the simulator paradigms. An advantage from one paradigm can be used to replace a drawback in another paradigm to form a hybrid.

4.1.2 Rigid bodies

A rigid body is a solid body that can not be deformed. This is, any particles in the rigid body always have the same distance between them. A rigid body's center of mass can both be translated and rotated, thus it have 6 degrees of freedom. The properties of a rigid body is determined from its inertia tensor and mass. The inertia tensor is the rotational equivalence of the total mass. It contains information about the body's mass distribution and is used for determining the body's rotation velocity around any given axis [24].

Equations of motion for a rigid body

From the Newton-Euler equations of motion,

$$\begin{array}{lcl} m\,\dot{\mathbf{v}}_{cm} & = & \mathbf{f} \\ \dot{\mathbf{x}}_{cm} & = & \mathbf{v}_{cm} \\ \mathbf{I}\,\dot{\boldsymbol{\omega}}_{cm} & = & \boldsymbol{\tau}_{cm} + \boldsymbol{\omega} \times \mathbf{I}\,\boldsymbol{\omega} \\ \dot{q} & = & \frac{1}{2}\,\boldsymbol{\omega}\,q \end{array}, \tag{4.4}$$

the differential equation for integrating the rigid body's motion is derived

$$\frac{d}{dt}\begin{bmatrix}\mathbf{x}_{cm} \\ q \\ \mathbf{v}_{cm} \\ \boldsymbol{\omega}\end{bmatrix} = \begin{bmatrix}\mathbf{v}_{cm} \\ \frac{1}{2}\,\boldsymbol{\omega}\,q \\ m^{-1}\,\mathbf{f} \\ \mathbf{I}^{-1}(\,\boldsymbol{\tau}_{cm} + \boldsymbol{\omega} \times \mathbf{I}\,\boldsymbol{\omega}\,)\end{bmatrix}. \tag{4.5}$$

The dot notation corresponds to the derivative of the variable. Here the quaternion q is the body's orientation, $\mathbf{x}_{cm}$ and $\mathbf{v}_{cm}$ is the position and velocity respectively of the rigid body's center of mass. Forth $\boldsymbol{\omega}$ is the rotational velocity, $\boldsymbol{\tau}_{cm}$ is the torque, $\mathbf{f}$ is the total force acting on the body's center of mass, m is the mass of the body and I is its inertia tensor. The subscript cm means that parameters are applied to the body's center of mass. The formulation in equation 4.5 is called the Lagrangian formulation [24].

4.1.3 Time integration

The motion of a rigid body's center of mass can be described as an ordinary differential equation (ODE) on the form

$$\dot{\mathbf{Y}} = f(\mathbf{Y}), \tag{4.6}$$

where $\mathbf{Y}$ is the body's state, including its position, velocity, orientation and rotational velocity. If time is discretized, Taylors theorem gives

$$\begin{array}{rcl} \mathbf{Y}(t+h) & = & \mathbf{Y}(t) + \dot{\mathbf{Y}}(t)\,h + \frac{1}{2}\,\ddot{\mathbf{Y}}(t)\,h^2 + \dots \\ \Rightarrow\ \mathbf{Y}(t+h) & = & \mathbf{Y}(t) + \mathbf{f}_t\,h\, + O(h^2) \end{array} \tag{4.7}$$

where t is the current time, h is the size of the time step, $\mathbf{f}_t$ is the force acting on the body's center of mass at time t and $O(h^2)$ is the rest of the terms in the Taylor expansion which can be disregarded for small time steps. From equations 4.7 and 4.5 Euler's method for numerically solving ODEs can be derived and used for integrating the motion of the body. From initial conditions for the position $\mathbf{x}_0$ and the velocity $\mathbf{v}_0$ Euler's method calculate the position and velocity for the next point in time, $t+h$, from the acceleration $\mathbf{a} = \mathbf{f}\,m^{-1}$ [27].

Explicit Euler

$$\begin{array}{rcl} \mathbf{x}_{t+h} & = & \mathbf{x}_t + h\mathbf{v}_t \\ \mathbf{v}_{t+h} & = & \mathbf{v}_t + h\mathbf{a}_t \end{array}, \tag{4.8}$$

The algorithm calculates the new velocity, $\mathbf{v}_{t+h}$, from the acceleration, $\mathbf{a}_t$, at current time t and the new position, $\mathbf{x}_{t+h}$, from the velocity at current time t. Forth h is the size of the time step. Explicit Euler has an error of size $O(h^2)$ that is accumulated in each iteration. For large time step h it is unstable and not very accurate. Explicit Euler should be avoided in multibody dynamic simulations [24].

Implicit Euler

$$\begin{array}{rcl} \mathbf{x}_{t+h} & = & \mathbf{x}_t + h\mathbf{v}_{t+h} \\ \mathbf{v}_{t+h} & = & \mathbf{v}_t + h\mathbf{a}_{t+h} \end{array}. \tag{4.9}$$

The new velocity, $\mathbf{v}_{t+h}$, is calculated from the acceleration, $\mathbf{a}_{t+h}$, at time $t + h$ and the new position, $\mathbf{x}_{t+h}$, is calculated from the velocity at time $t + h$. The size of the time step is denoted with h. This gives a system of coupled equations. Implicit Euler is stable regardless the size of h, but it is dissipative which is usually not good in simulation of multibody dynamics [24, 27].

Symplectic Euler

$$\begin{array}{rcl} \mathbf{x}_{t+h} & = & \mathbf{x}_t + h\mathbf{v}_{t+h} \\ \mathbf{v}_{t+h} & = & \mathbf{v}_t + h\mathbf{a}_t \end{array}, \tag{4.10}$$

In symplectic Euler the new velocity, $\mathbf{v}_{t+h}$, is calculated from the acceleration, $\mathbf{a}_t$, at time t while the new position, $\mathbf{x}_{t+h}$, is calculated from the velocity at time $t + 1$. Forth h is the size of the time step. This algorithm has good stability in a wide parameter range and suits very well for multibody dynamic simulations [27].

4.1.4 Constraints

As established in section 4.1.2 a rigid body have 6 degrees of freedom. Consider two rigid bodies, i and j, which together have 12 degrees of freedom, these bodies can then be articulated into joints by removing degrees of freedom between the two bodies. The removing of degrees of freedom is done with time independent equality constraints, which are called bilateral holonomic constraints. The two bodies position and orientation form the generalized position vector $\mathbf{s} = \left[\mathbf{x}_i, q_i, \mathbf{x}_j, q_j\right]$, the constraint

$$\begin{array}{rcl} \Phi_1(\mathbf{s}) & = & 0 \\ \Phi_2(\mathbf{s}) & = & 0 \\ & \vdots & \\ \Phi_m(\mathbf{s}) & = & 0 \end{array}, \tag{4.11}$$

removes m degrees of freedom between bodies i and j.

For example, a ball and socket joint removes all translational degrees of freedom between the bodies i and j, this is shown in equation 4.12.

$$\begin{array}{rcccl} \Phi_1(\mathbf{s}) & = & x_i - x_j & = & 0 \\ \Phi_2(\mathbf{s}) & = & y_i - y_j & = & 0 \\ \Phi_3(\mathbf{s}) & = & z_i - z_j & = & 0 \end{array}, \tag{4.12}$$

where x, y and z are the components of a body's position vector $\mathbf{x}$.

A holonomic constraint can be differentiated into a kinematic constraint,

$$\frac{d}{dt}\Phi(\mathbf{s}) = \frac{\partial \Phi}{\partial \mathbf{s}} \frac{d\,\mathbf{s}}{dt} = \frac{\partial\, \Phi}{\partial\, \mathbf{s}} \mathbf{S}\,\mathbf{u} = \mathbf{J}_{\Phi}\,\mathbf{u}, \tag{4.13}$$

where $\mathbf{u} = \left[\mathbf{v}_i, \omega_i, \mathbf{v}_j, \omega_j\right]$, $\mathbf{S}$ is a transformation matrix that transforms $\mathbf{u}$ into the derivative, $\dot{\mathbf{s}}$, of the $\mathbf{s}$ vector. $\mathbf{J}_{\Phi} = \frac{\partial\,\Phi}{\partial\,\mathbf{s}}\,\mathbf{S} \in \mathbb{R}^{mx12}$ is the Jacobian matrix with m number of holonomic constraints. The Jacobian matrix describes how the degrees of freedom are reduced [27]. From the principle of virtual work it follows that the constraint forces can be written as

$$\mathbf{f}_{\boldsymbol{\Phi}} = \mathbf{J}_{\Phi}^{T}\,\boldsymbol{\lambda}_{\Phi}, \tag{4.14}$$

where $\boldsymbol{\lambda}_{\Phi} \in \mathbb{R}^{m}$ is a vector of Lagrange multipliers and the superscript ^{T} denotes the transpose [24].

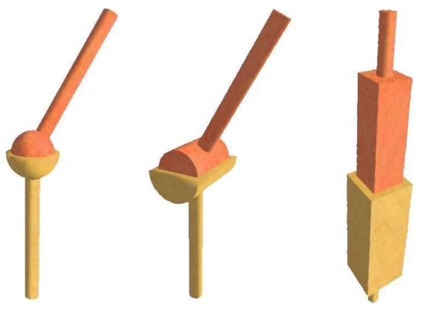


Figure 4.1: Three different types of joints, from left: ball and socket joint, hinge (revolute) joint and prismatic (translational) joint. Figure from [6].

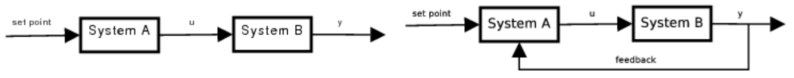
Joint types

Three common joint types are ball and socket joint, hinge joint and prismatic joint. The ball and socket joint is previously described, an illustration of the ball and socket constraint is shown to the left in figure 4.1. A hinge, or revolute, joint allows a relative rotation between the bodies around a joint axis. The hinge joint is illustrated in the middle of figure 4.1. The prismatic, or translational, joint is illustrated to the right in figure 4.1. It allows a relative translation between the bodies along a joint axis.

4.2 Control theory

A dynamical system is a system that change its behavior over time, often due to external manipulations. Control theory is the theory of controlling the behavior of a dynamical system. A control system is a system that controls another system. A system can consist of one or several devices. In this case *automatic* control systems are considered. The term "control system" can also be applied to systems that need manual control to control another system. For example, an operator use a control system for opening and closing a valve in a hydraulic system.

If system A is a controller for system B, then if system A is provided with feedback from the output of system B, the total system is called a closed loop system. If there are no feedback from the output of system B, the total system is called an open loop system [7], this is illustrated in figure 4.2.



(a) Open loop. (b) Closed loop.

Figure 4.2: Open loop and closed loop systems.

For a closed loop system the feedback from the output can be used to compensate for controlling errors. In an open loop system the value of the output can not influence the control signal. Most control systems is closed loop systems. A system with feedback can correct for external disturbances and variations in it's internal components. Feedback also introduce some disadvantages, it may cause oscillations or infinite values in the system output. The use of feedback requires sensors for the feedback term, the sensors can then introduce noise in the system. Therefore it's important to filter these signals.

Some classical control principles are: On-off control, Proportional control, Integral control, PI control, PD control and PID control. This thesis will only consider PID control, see [28, 7] for the other principles.

4.2.1 PID control

PID control is a technique to control a parameter in a system or a process. For example in a temperature regulating system the controlled parameter is the temperature, in a speed control system for a car, the speed is the controlled parameter. PID is acronym for Proportional, Integral and Derivative. These terms refer to how the parameter is controlled, the terms are e further described in this section.

The reference value (or set point), denoted r, is the desired value on the controlled parameter. The current value of the controlled parameter is denoted y and is in this text mentioned as system output. The system output is measured with a sensor. The difference $r - y$ constitute the control error denoted e. From the control error e the PID control algorithm calculates a control signal u for the actuator that attempt to regulate the controlled parameter towards the reference value r.

The control law for a basic PID controller can be expressed mathematically [7] as

$$u(t) = u_p(t) + u_i(t) + u_d(t) = k_p e(t) + k_i \int_0^t e(t)\,\mathrm{d}t + k_d \frac{\mathrm{d}e(t)}{\mathrm{d}t}, \qquad (4.15)$$

where $e(t)$, k_p, k_i, k_d is the error at time t, the proportional gain, integral gain and derivative gain respectively. The gains are constants while the error is calculated as the difference between the reference value and the system output, $r(t) - y(t)$, at time t. This control law consider the error from the past, with the integral term, the present, with the proportional term, and the future with the derivative term.

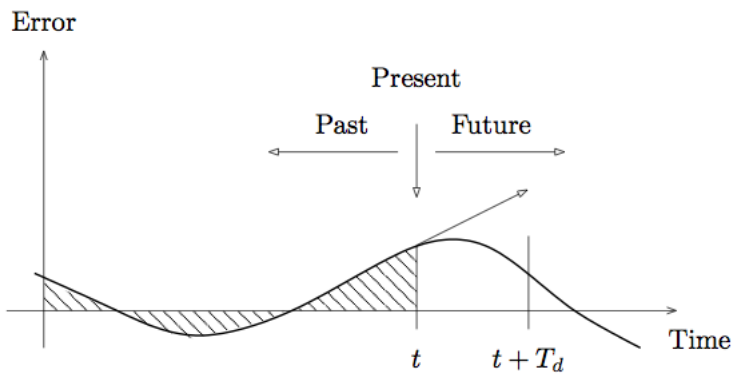


Figure 4.3: PID control consider the past, present and future. Figure from [7].

Proportional term

The proportional term in equation 4.15

$$u_p(t) = k_p e(t) = k_p(r(t) - y(t)) \tag{4.16}$$

makes a compensation in the controlled parameter proportional to the error. The error is $r(t)$ - $y(t)$ at time t, where $r(t)$ and $y(t)$ is the reference and system output respectively at time t. The constant k_p is a gain acting on the error term. A small gain will result in a stable but slow system. If the gain is increased the system will respond faster, but with to high gain it will be unstable. With pure proportional control there is no guarantee for the controlled parameter to reach the reference value [7, 28].

Integral term

In equation 4.15 the integral term

$$u_i(t) = k_i \int_0^t e(t)\,\mathrm{d}t, \tag{4.17}$$

contributes to the control signal with the integral of the error $e(t)$. A positive error will increase the integral part, while a negative error will decrease it. The constant k_i is the integral gain, which corresponds to the integral contribution in the total control signal. Due to the integral term a zero steady state error can be achieved [7, 28].

Derivative term

The derivative term in equation 4.15

$$u_d(t) = k_d \frac{\mathrm{d}e(t)}{\mathrm{d}t}, \tag{4.18}$$

tries to predict the error by the derivative of the error $e(t)$. The derivative term will contribute to the total signal only when the change in $e(t)$ differ from zero. When the error increase the derivative term will have the same sign as the proportional term, but when the error decrease it will have the opposite sign to the proportional term. The derivative term contribute with fast stabilization on disturbance, neutralize overshooting and instability. But it is sensitive for high frequent disturbance [7], [28].

4.2.2 Integral windup

The windup problem occurs with all integral controllers and comes from limitations in the controlled actuator. If the controller output transcends a limit of the actuator control signal the actuator will work at its limit and can not achieve the set point. This cause an error between system output and the set point that is impossible for the controller to compensate for. Which in turn cause a growing magnitude of the integral term, which continue to accumulate from the error. The magnitude of the integral term and the controller output can therefore be very large and still remain saturated when the error changes. It can take long time before the integral term and controller output come inside the saturation range again.

In [7] integrator windup is avoided by an extra feedback term that is constructed from the difference between the controller output and the actuator output. Integrator windup protection can easily be implemented with the differential PID control algorithm, see section 4.2.5.

4.2.3 Derivative filtering

High frequent noise cause major problems in the derivative term because of its high amplitude gain for high frequencies. This results in large, unstructured and fast variations in the control signal. This can be solved by a low pass filter on the derivative term. A low pass filter only allows frequencies below a certain threshold. Another solution is a low pass filter on the controller input signal [7] [28].

4.2.4 Discrete PID control

A computer is digital and have to convert everything thats in analog into a digital format. The continuous PID control law have to be approximated with a discrete version if it should be implemented in a computer. Equation 4.19 is an approximated discrete version of equation 4.15.

$$u(t) = u_p(t) + u_i(t) + u_d(t) = k_p e(t) + hk_i \sum_{i=1}^{t} e(i) + k_d \frac{e(t) - e(t-1)}{h}, \tag{4.19}$$

where the constants are the same as in equation 4.15, t is the current time and h is the size of the time step.

4.2.5 Discrete differential PID control

Equation 4.19 can be rewritten to

$$u(t) = u(t-1) + \Delta u, \tag{4.20}$$

where

$$\Delta u = k_p(e(t) - e(t-1)) + k_i e(t) + k_d(e(t) + e(t-2)). \tag{4.21}$$

This is the differential form of the discrete PID control law. The differential form have some advantages over ordinary PID control, it is easier to implement protection against integral windup and protection against bumps when the PID parameters are changed online. But the differential control law does not work without integral action, then it can not keep a stationary system. In equation 4.22 the controller output signal is clamped between min and max, which are the limits of the actuators control signal. This clamp gives protection against integral windup [29].

$$u(t) = (u(t-1) + \Delta u)|_{min}^{max}, \tag{4.22}$$

4.3 Forward and inverse kinematics

In robotics a manipulator consists of several links connected by joints to form a kinematic chain [30]. The links are rigid bodies and a joint is a constraint between to two links. There are several types of joints, but here the focus will be on revolute and prismatic joints. For a revolute joint the attached bodies are allowed to rotate relative to each other around an axis. For a revolute joint the rotation angle around the rotation axis between the two participating bodies is denoted θ. For the prismatic joint the bodies are allowed to move relative to each other along an axis. For prismatic joints the relative displacement between the participating bodies along the joint axis is denoted d. θ and d are called joint variables. A device at the end of the robot manipulator is called end effector.

If a manipulator configuration can be minimally specified by n parameters it is said to have n degrees of freedom. The number of joints is the degree of freedom for a robotic manipulator. If a manipulator have more degrees of freedom than necessary to accomplish its tasks, it is said to be a redundant manipulator.

This thesis will look closer to a forwarder crane, which have 4 degrees of freedom, figure 4.4 shows the kinematics of a typical forwarder crane. Joint variables $\theta_1 - \theta_3$ are revolute joints while the fourth joint is prismatic and denoted by d. The fact that it have four joints but operates in 3 dimensions makes this manipulator redundant.

For the manipulator in figure 4.4 three kinematic problems are of interest for this thesis, to find end effector position given the positions of the joints, find the end effector velocity given the velocities of the joints and find velocities for the joints given the velocity of the end effector. The first two is called forward or direct kinematics for position and velocity, this problem is solved with simple trigonometry in the position case. The third one is called inverse kinematics and is in general more difficult, especially with a redundant manipulator. A redundant manipulator typically have infinitely many solutions to the inverse problem.

In this text the assignment of frames to each link and the general transformation between the frames are not considered, the focus will be on the forwarder crane only. For more information of a general case, see [31]. This section will introduce forward kinematics for both position and velocity and inverse kinematics for velocity. This is done by deriving kinematic equations for the crane in figure 4.4, from which the position of the end effector can be calculated. To be able to calculate the end effector velocity the Jacobian matrix is derived from the kinematic equations. The Jacobian matrix is then used to solve the inverse velocity problem.

4.3.1 Forward kinematics

Given the position or velocity for each joint of a manipulator the position or velocity of the end effector should be calculated. For a forwarder crane the boom tip is the manipulator end effector. To find the position of the boom tip is a trivial problem when

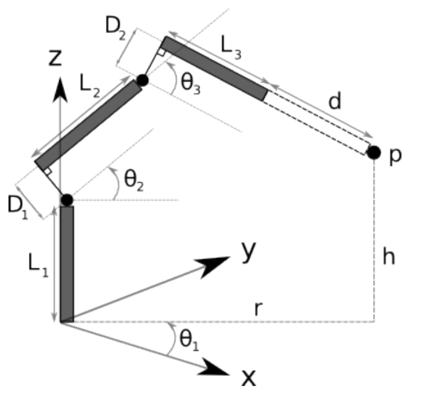


Figure 4.4: Kinematics of a typical forwarder crane.

the dimensions of the crane is known. With the notations used in figure 4.4 the boom tip position $\mathbf{p}$ is a function of the joint vector $\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3, d]^T$

$$\mathbf{p} = f(\boldsymbol{\theta}), \tag{4.23}$$

where f is a function from joint space to Cartesian space

$$f(\boldsymbol{\theta}) = \left[p_x, p_y, p_z\right]^T = \left[cos\,\theta_1\, r,\ sin\,\theta_1\, r,\ h\right]^T, \tag{4.24}$$

where the extension of the boom tip, r, and its height, h, are

$$\begin{array}{l} r = L_2\, cos\,\theta_2 - D_1\, sin\,\theta_2 + (L_3 + d)\, cos\,(\theta_2 + \theta_3) - D_2\, sin\,(\theta_2 + \theta_3) \\ h = L_1 + L_2\, sin\,\theta_2 + D_1\, cos\,\theta_2 + (L_3 + d)\, sin(\theta_2 + \theta_3) + D_2\, sin(\theta_2 + \theta_3) \end{array}.$$

The superscript ^{T} denotes the transpose.

The relation between joint velocities and the boom tip velocity are described by the Jacobian relationship

$$\dot{\mathbf{p}} = \mathbf{J}\,\dot{\boldsymbol{\theta}} \tag{4.25}$$

where $\dot{\mathbf{p}}$ is the unknown 3×1 velocity vector for the boom tip, $\dot{\boldsymbol{\theta}} = [\ \dot{\theta_1},\ \dot{\theta_2},\ \dot{\theta_3},\ \dot{d}\ ]^T$ is the 4×1 joint velocity vector and $\mathbf{J}$ is the 3×4 Jacobian. The Jacobian matrix is a function of the joint position vector $\boldsymbol{\theta}$, its column entries for the i:th column are determined from

$$\begin{array}{ll} \mathbf{J}_i = \dfrac{\partial\,\mathbf{p}}{\partial\,\theta_i} = \mathbf{v}_i, & \text{if joint } i \text{ is prismatic} \\ \mathbf{J}_i = \dfrac{\partial\,\mathbf{p}}{\partial\,\theta_i} = \mathbf{v}_i \times (\mathbf{p} - \mathbf{p}_i), & \text{if joint } i \text{ is revolute} \end{array}, \tag{4.26}$$

where $\mathbf{p}_i$ is the position of joint i, $\mathbf{p}$ is the boom tip position and $\mathbf{v}_i$ is a unit vector along the joint axis for joint i. From figure 4.4 we derive the Jacobian for the forwarder crane as

$$\mathbf{J} = \left[(\mathbf{v}_1 \times (\mathbf{p} - \mathbf{p}_1))^T,\ (\mathbf{v}_2 \times (\mathbf{p} - \mathbf{p}_2))^T,\ (\mathbf{v}_3 \times (\mathbf{p} - \mathbf{p}_3))^T,\ \mathbf{v}_4^T\right] \tag{4.27}$$

The joint axises are

$$\begin{array}{l} \mathbf{v}_1 = [\,0 \quad 0 \quad 1\,]^T \\ \mathbf{v}_2 = \mathbf{v}_3 = [\ sin\,\theta_1 \quad -cos\,\theta_1 \quad 0\,]^T \\ \mathbf{v}_4 = [\ cos\,\theta_1 \quad sin\theta_1 \quad sin\,(\theta_2 - \theta_3)\ ]^T \end{array}, \tag{4.28}$$

and the joint positions for the revolute joints are

$$\begin{aligned}
\mathbf{p}_1 &= [\,0 \quad 0 \quad 0\,]^T \\
\mathbf{p}_2 &= [\,0 \quad 0 \quad L_1\,]^T \\
\mathbf{p}_3 &= [\,cos\,\theta_1\, r_{p_3},\; sin\,\theta_1\, r_{p_3},\; h_{p_3}\,]^T
\end{aligned} \tag{4.29}$$

where

$$\begin{aligned}
r_{p_3} &= L_2\; cos\,\theta_2 - D_1\; sin\,\theta_2 \\
h_{p_3} &= L_1 + L_2\; sin\,\theta_2 + D_1\; cos\,\theta_2
\end{aligned},$$

are the extension and height respectively for $\mathbf{p}_3$. L_1, L_2, D_1 and D_2 are defined in figure 4.4.

The end effector position $\mathbf{p}$ is calculated with equation 4.23. From equations 4.23, 4.27, 4.28 and 4.29 the complete Jacobian matrix is formed

$$\mathbf{J} = \begin{bmatrix} -s_{\theta_1}\,\lambda & -c_{\theta_1}\,\gamma & -c_{\theta_1}\,\vartheta & c_{\theta_1} \\ c_{\theta_1}\,\lambda & -s_{\theta_1}\,\gamma & -s_{\theta_1}\,\vartheta & s_{\theta_1} \\ 0 & {s_{\theta_1}}^2\,\lambda + {c_{\theta_1}}^2\,\lambda & s_{\theta_1}\,(\,s_{\theta_1}\,\lambda - s_{\theta_1}\,\zeta\,) + c_{\theta_1}(\,c_{\theta_1}\,\lambda - c_{\theta_1}\,\zeta\,) & s_{\theta_2-\theta_3} \end{bmatrix} \tag{4.30}$$

where

$$\begin{aligned}
s_x &= sin(\,x\,), \quad x \in \mathbb{R} \\
c_x &= cos(\,x\,), \quad x \in \mathbb{R} \\
\vartheta &= (\,L_3 + d\,)\; sin\,(\theta_2 + \theta_3) + D_2\; sin(\theta_2 + \theta_3) \\
\gamma &= L_2\; sin(\theta_2) + D_1\; cos(\theta_2) + \vartheta \\
\zeta &= (L_2\; cos\,(\theta_2) - D_1\; sin\,(\theta_2) \\
\lambda &= \zeta + (\,L_3 + d\,)\; cos\,(\theta_2 + \theta_3) - D_2\; sin\,(\theta_2 + \theta_3)
\end{aligned}.$$

4.3.2 Inverse kinematics

A more challenging problem is that of find the joint velocities from a given end effector velocity. There can be several solutions to this problem and with an redundant manipulator there is typically an infinite number of solutions to any given end effector position. The forwarder crane in figure 4.4 is a redundant manipulator because of the prismatic joint d.

Equation 4.25 describes the velocity of the boom tip, $\dot{\mathbf{p}}$, in terms of the joint velocity vector $\dot{\boldsymbol{\theta}}$. For the inverse problem, joint velocities in terms of the boom tip velocity are desired

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{-1}\,\dot{\mathbf{p}}. \tag{4.31}$$

But the Jacobian matrix $\mathbf{J}$ is not square for a redundant manipulator, and hence, not invertible. There are several methods to solve the problem with the non invertible Jacobian [32], this thesis will look at a pseudo inverse method [5, 10, 17, 32].

If equation 4.31 is solved with

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{\dagger}\,\dot{\mathbf{p}}, \tag{4.32}$$

where $\mathbf{J}^{\dagger}$ is the Moore-Penrose pseudo inverse Jacobian matrix

$$\mathbf{J}^{\dagger} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}. \tag{4.33}$$

Equation 4.33 gives the best possible solution to equation 4.31 by minimize a cost function [32]

$$C = \dot{\boldsymbol{\theta}}^T\,\dot{\boldsymbol{\theta}} = \dot{\theta}_1^2 + \dot{\theta}_2^2 + \dot{\theta}_3^2 + \dot{d}^2. \tag{4.34}$$

$\mathbf{J}^\dagger$ is defined for all matrices $\mathbf{J}$. In the cost function in equation 4.34 the three revolute joints are treated as if they have the same dimension as the prismatic joint. This is not the case, revolute joints are measured in radians while prismatic joints are measured in meters. Therefore a positive definite weighting matrix is introduced [17]

$$\mathbf{W} = \begin{bmatrix} w_1 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & w_3 & 0 \\ 0 & 0 & 0 & w_4 \end{bmatrix}, \tag{4.35}$$

where w_i, $i = 1, 2, 3, 4$, are weights for the joints. By weighting the joints dimensions inequalities between the joints can be compensated and some joints can be given higher priority than other joints.

If the weighting matrix in equation 4.35 is included into equation 4.33 the result is a pseudo inverse Jacobian,

$$\hat{\mathbf{J}}^\dagger = \mathbf{W}\,\mathbf{J}^T\,[\mathbf{J}\,\mathbf{W}\,\mathbf{J}^T]^{-1}, \tag{4.36}$$

that will solve equation 4.32 with respect to the joint weights. This results in the following equation

$$\dot{\boldsymbol{\theta}} = \hat{\mathbf{J}}^\dagger\,\dot{\mathbf{p}}. \tag{4.37}$$

The matrix $\mathbf{J}\,\mathbf{W}\,\mathbf{J}^T$ in equation 4.36 will always be a $n \times n$ matrix where n is the number of manipulator joints. In the case with the forwarder crane the matrix is 3×3, its inverse can be expressed analytically by its elements. Because of the small matrix size the analytical inverse is suitable for real time computations.

Chapter 5

The `ATVS` framework

`ATVS` is an object oriented framework written in `C++` with functionality to control mechanical systems. Basically `ATVS` handles sensors, actuators, controllers and interaction with the user. With a sensor and an actuator for a system parameter it is possible to control the parameter with a controller. For example, if the angle in a hinge joint should be controlled, the angle is the system parameter. So if a sensor for measuring the angle and an actuator that can change the angle are supplied, the angle can be controlled by a controller. In `ATVS` a PID control algorithm is supplied and can be used in a controller. Sensors, actuators and controllers all have to be registered with a simulation which keep track of updating them each simulation step. General joystick and force feedback devices used in computer games are supported for interaction possibilities with the user.

A virtual environment is supplied in `ATVS` in which mechanical systems can be simulated. The virtual environment is developed with AgX multiphysics engine [1] and OpenSceneGraph [2]. Sensors and Actuators are created around joint constraints in the physics engine which makes it possible to control a joint in the virtual environment. With this functionality it is possible to create mechanical systems in the virtual environment and then provide sensors and actuators for the individual joints in the system. Development and testing of control algorithms that includes interaction with an operator can then easily be done. As an example this thesis have created a Valmet forwarder in the virtual environment and implemented a control algorithm to control the crane. Figures 1.2 and 5.5 illustrates the forwarder in the virtual environment.

If the framework is extend with implementations that can read values from real sensors and send signals to real actuators, a real system and a system simulated in the virtual environment can be controlled with the same controller, given that the kinematics of the systems are identical and the actuators response are the same. Otherwise the controller have to be tuned for the new system, but the same algorithm can still be used. The controller is supplied with sensors and actuators for the real system instead of the simulated system, but the controller will not notice any difference.

The purpose of `ATVS` is to provide tools where it is easy to experiment with control algorithms and investigate human machine interfaces for semi-autonomous mechanical systems with shared control and force feedback in virtual environments. The `ATVS` framework is a result of this master thesis. The development have mainly been done on the *Linux* platform in a platform independent way. The code is also compiled for the *Windows* platform with *Visual studio 2008*. To generate build files for different platforms the *CMake* [33] software is used. At the moment force feedback is only supported for the *Windows* platform. This chapter describes the main structure of `ATVS` and some implementation details.

5.1 System description

The `ATVS` framework have an abstract layer in the core `atvs` library. It includes interfaces for sensors, actuators and controllers, functionality to update implementations of these interfaces each simulation step and support for joystick and force feedback devices. It also contains support for defining tasks that can be executed in the simulation loop and an implementation of a differential PID control algorithm.

The library `atvsVE` contains the implementation of the virtual environment. Here the interfaces in the `atvs` library are implemented to create sensors and actuators. As an example figure 5.6 shows an inheritance diagram that shows how the sensor interface is implemented to create sensors in the virtual environment. The term *interface* should be theoretically interpreted. `C++` do not have support for real *interfaces* in the same way as `Java`. But similar functionality can be achieved with abstract base classes in `C++`.

The `atvs::Simulation` class is responsible for the updates of sensors, input system, tasks, controllers and actuators. A detailed description of the class is given in section 5.1.1. Tasks are handled by the task manager which is described in section 5.1.3. Joystick and force feedback support in `ATVS` are discussed in 5.1.4. Section 5.1.5 sum up utilities available for math in `ATVS`. In 5.1.6 memory management with reference counters are covered.

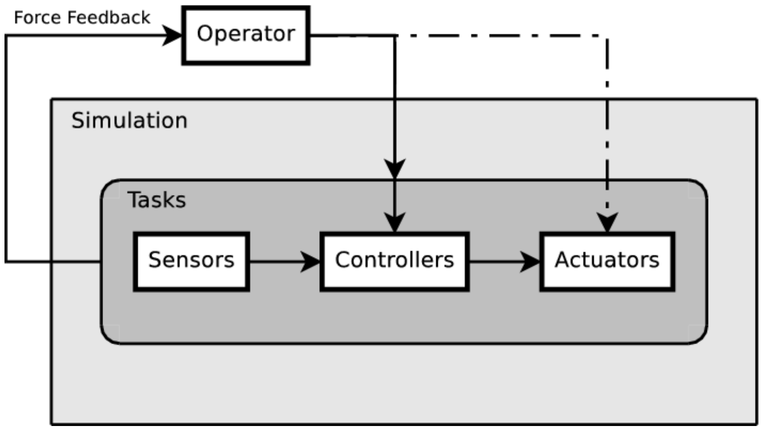


Figure 5.1: The main dataflow in the system.

5.1.1 The simulation class

Figure 5.1 shows a simplified view of the main flow of data in the simulation. Sensors, controllers and actuators can be included in Tasks, which typically implements control algorithms on a higher level. The dataflow from the operator can go either directly to the controllers or via a task for processing before it reaches the controllers. The operator can also directly control the actuators. The controllers are supplied with data from sensors, other controllers and the operator. Typically the controllers implements a control algorithm that calculate control data for the actuators. Force feedback is generated back to the operator to get a bilateral control architecture.

The simulation class is the heart in the `ATVS` framework. Each sensor, controller and actuator that should be a part of the simulation must be registered with the simulation class. Tasks are added to a task library in the task manager, read more about the task manager in section 5.1.3. It is possible to add subsystems that implements the `atvs::SubSystem` interface to the simulation class. The task manager implements the sub system interface and is registered with the simulation as a subsystem. The simulation class also have to update the input system each simulation step, read more about the input system in section 5.1.4. In each simulation step the simulation class updates the objects registered with it in the following way:

1. **Sensors** – Read values from all sensors.
2. **Input system** – Update the input system, which extracts data from input devices.
3. **Subsystems** – Update registered subsystems. The task manager is ordered to update its tasks here.
4. **Controllers** – Let the controllers calculate the control signals.
5. **Actuators** – Update the actuators with the new control signals.

The simulation class for the virtual environment

In the virtual environment the dynamical simulation is controlled by the AgX physics engine [1]. AgX provide rendering with OpenSceneGraph [2], which is used by `ATVS`. To cooperate with the dynamic- and rendering simulation in the virtual environment, `atvs::Simulation` is extended in the `atvsVE` library and named `atvsVE::Simulation`. The `atvsVE::Simulation` class gets a notification to take a simulation step before each simulation step in the dynamic simulation is executed. The update of the rendering is controlled in the dynamic simulation by AgX. Adding of rigid bodies, constraints and graphical models to AgX and OpenSceneGraph are also handled by `atvsVE::Simulation`.

5.1.2 Simulation objects

Simulation objects are containers for sensors, controllers and actuators that are related to each other. For example sensors, controllers and actuators of a mechanical system can be stored in a simulation object. In this way all information about a mechanical system can be stored in a single object and it will be easier to handle. In the virtual environment also rigid bodies and constraints can be associated with a simulation object.

5.1.3 The task manager

The task manager handles tasks in `ATVS`. A task typically implements control algorithms on a high level that includes several controllers implementing low level control algorithms. A task manager have its own task library where it store tasks that is associated with the task manager. The tasks are identified with their names, they are stored in a map in the library with the name as look up key. The task manager have methods for adding tasks to the library, fetch tasks from the library, start execution of tasks, stop tasks etc. A collaboration diagram for the task manager is provided in figure 5.2. When the execution of a task is started, the task is put in an initializing

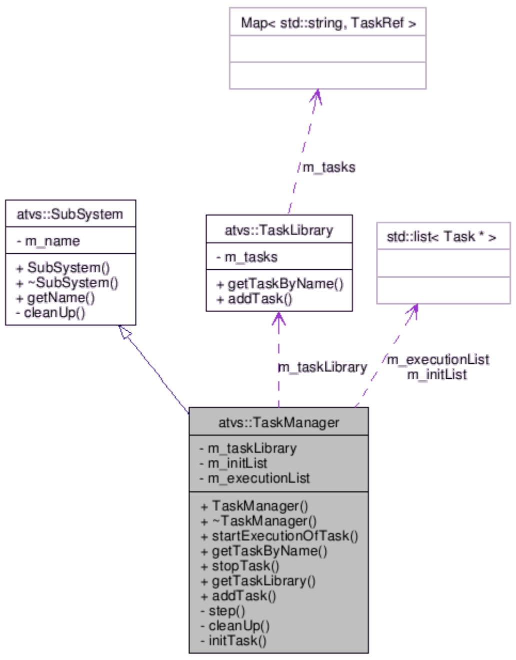


Figure 5.2: Collaboration diagram for the task manager.

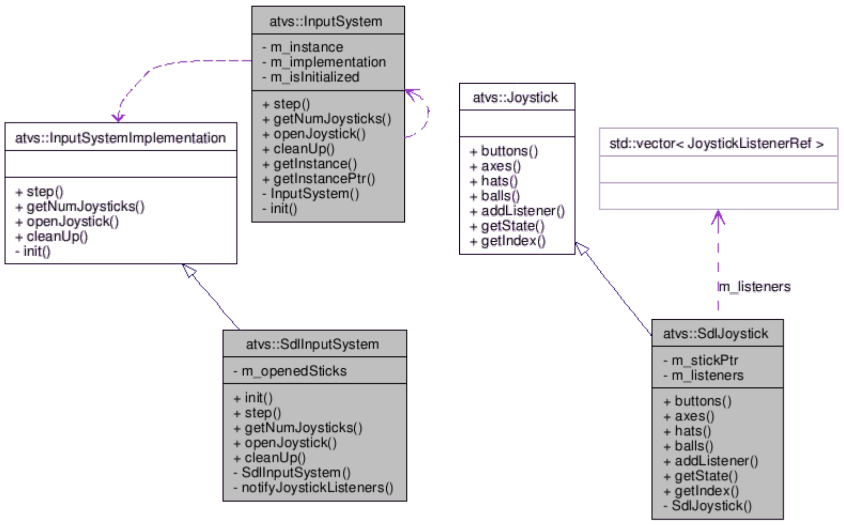
phase that is executed in a separate thread. When the initialization phase is done, the thread is terminated and the task is put into an execution list in the task manager. The task manager then updates the tasks in the execution list each simulation step. If a task reach its stop criteria and is done, it is removed from the execution list. The task manager implements the sub system interface and is registered with the simulation class as a sub system. This means that the task manger get an update notification after the sensors and input system is updated. The task can then be provided with new data from sensors and operator.

5.1.4 Joystick and force feedback devices

The `atvs` library contains high level interfaces to receive inputs from general gaming joysticks and actuate constant forces in general gaming force feedback devices. In figure 5.4 a joystick with force feedback support is shown. The implementations of these interfaces are done with the help of existing software libraries, such as *DirectX* and *Simple DirectMedia Layer*.

Joysticks

Input devices are handled by the singleton class `atvs::InputSystem`. Currently it only supports joystick devices. The class wraps an implementation, `atvs::InputSystemImplementation`, that handles the actual functionality of finding input devices and receiving inputs from them. The fact that the implementation is wrapped into another class makes it possible to have different implementations for different platforms and still provide the same interface to the user. Consequently the user is



(a) SDL based implementation of the input system.

(b) SDL based joystick implementation.

Figure 5.3: Collaboration diagrams for the SDL implementations of the input system and the joystick.

unaware of the underlying implementation. At the moment there exists one implementa-

Figure 5.4: A gaming joystick with force feedback support from Saitek.

tion based on the *Simple DirectMedia Layer* (SDL) for handling joystick communication. SDL is available to both Linux and Windows, so there is no need to have different implementations for the two operating systems. Figure 5.3 shows a collaboration diagram for the classes included in the SDL input system implementation.

An interface for joysticks, `atvs::Joystick`, makes it possible to get information about a joystick device and also add listeners to the joystick. The interface methods must be implemented by implementation specific joystick classes. Figure 5.3b shows how the SDL specific implementation for joysticks inherit from the `atvs::Joystick` interface. To receive input information from a joystick a listener have to be added to the joystick. Joystick listeners have to implement the interface `atvs::JoystickListener` which contains methods that are called when there is information available from the joystick, for example when an axis is moved or when a button is pressed.

Force feedback devices

The support for force feedback devices follows the same structure as for joysticks, see figure 5.3 and section 5.1.4. The class `atvs::ForceFeedbackSystem` is a singleton that wraps the actual implementation which handles the force feedback devices. An interface for a force feedback device, `atvs::ForceFeedbackDevice`, makes it possible to set constant forces on the x and y axises of the device. Of course it must be a device with support for constant forces on the x and y axises.

Simple DirectMedia Layer (SDL), which is used in the input system implementation, currently lack support for force feedback devices. But in the next version (1.3) of SDL there will probably be support for them. Therefore *DirectInput*, which is a part of Microsoft's *DirectX* library is used to implement the force feedback functionality. The fact that *DirectX* are used in many games with force feedback support and that it is rather well documented motivated the choice of *DirectInput*. Due to the use of *DirectInput* force feedback is only available in the Windows version of `ATVS`.

5.1.5 Math utilities

`ATVS` use open source code from the *OpenSceneGraph* project [2] to provide a collection of math functions and classes. There are classes for two, three and four dimensional vectors, a 4x4 matrix and some basic utility functions such as clamping values etc. The vector classes have implemented functionality for dot product, cross product, normalization, length, squared length and basic arithmetic. The matrix class have functionality for matrix multiplication, transpose, inverse and methods that easily make the matrix to a translation-, scale-, or rotation matrix. It also have methods create common matrices that appears in computer graphics, but these methods are not necessary for this project.

5.1.6 Memory management

To handle deallocation of allocated objects `ATVS` use open source code from the *OpenSceneGraph* project [2]. The deallocation is handled with reference counting. This is a technique that keeps track of how many pointers there are to an object, when the number of pointers decrease to zero the object will be deallocated. The reference counter is kept in the object it self, so all classes that should have support for reference counting should inherit a class that handles the reference counting functionality. When an object is associated with a reference pointer the reference counter in the object is increased with one, when the object is separated from a reference pointer the counter is decreased with one.

Below is a simple example function to illustrate how reference pointers work. A reference pointer, `exampleRef`, is associated with an allocated object of the fictional class `ExampleObject` and the reference counter in the object is increased to be one. When the function returns the reference pointer `exampleRef` goes out of scope and the reference counter in the object is decreased to zero and the object will deallocate itself. A more detail documentation on reference counting can be found in the *OpenSceneGraph* project [2].

```
void function()
{
    atvs::ref_ptr< atvs::ExampleObject > exampleRef = new ExampleObject;
}
```

5.2 Implementation details

In this section some details about the implementation is described. The PID control algorithm implemented in `ATVS` are described in section 5.2.1. Section 5.2.3 shows how the Valmet forwarder is modeled with rigid bodies and constraints. The implementation for velocity control of the boom tip on a forwarder crane is described in section 5.2.2. How sensors and actuators are created in the virtual environment is covered in sections 5.2.4 and 5.2.5

5.2.1 PID control implementation

Section 4.2.5 on page 22 discuss differential PID control.
A differential pid control algorithm with anti windup protection and derivative filter is implemented in `ATVS`. The ideal discrete form of the PID control algorithm

$$u(t) = u_p(t) + u_i(t) + u_d(t), \tag{5.1}$$

where

$$\begin{array}{rcl} u_p(t) &=& k_p\ e(t) \\ u_i(t) &=& u_i(t-1) + k_i\ h\ e(t-1) \\ u_d(t) &=& \frac{T_f}{T_f+h} u_d(t-1) - \frac{k_d}{T_f+h}(\ y(t) - y(t-1)\ ) \end{array},$$

is rewritten into its differential form

$$u(t) = u(t-1) + \varDelta u(t) = u(t-1) + \varDelta u_p(t) + \varDelta u_i(t) + \varDelta u_d(t) \tag{5.2}$$

where

$$\begin{array}{rcl} \varDelta u_p(t) &=& k_p(\ e(t) - e(t-1)\ ) \\ \varDelta u_i(t) &=& k_i\ h\ e(t-1) \\ \varDelta u_d(t) &=& \frac{T_f}{T_f+h}\ \varDelta u_d(t-1) - \frac{k_d}{T_f+h}(\ y(t) - 2y(t-1) + y(t-2)\ ) \end{array}.$$

h is the time step and $T_f = (k_d/k)/N$ is the filtering time, where N range from 2-20.

Integral windup protection is implemented by clamping $u(t-1) + \varDelta u(t)$ between the actuator control signal limits. Clamping means that if the value is at the wrong side of a limit, it is set to the value of the limit.

$$u(t) = (\ u(t-1) + \varDelta u(t)\ )\ |_{min}^{max} \tag{5.3}$$

This PID control algorithm is implemented in the class `atvs::PidControl`.

5.2.2 Implementation of velocity control for the boom tip

Here the implementation of velocity control, including joint limit protection, for the boom tip of a forwarder crane is presented. Sections 4.3.1 and 4.3.2 covers the relevant theory, which builds on [17, 10], for velocity control of the boom tip. Recall equation 4.37 from section 4.3.2 on page 26

$$\dot{\boldsymbol{\theta}} = \hat{\mathbf{J}}^{\dagger}\ \dot{\mathbf{p}},$$

which use the Moore–Penrose pseudo inverse Jacobian, $\hat{\mathbf{J}}^{\dagger}$, to find the best possible solution by minimizing the cost function in equation 4.34. Where $\dot{\boldsymbol{\theta}}$ is the unknown joint velocity vector and $\dot{\mathbf{p}}$ is the reference for the boom tip velocity. To avoid that joints are commanded to work outside the limits of their working range, they can be weighted down when they approaching a limit. The pseudo inverse Jacobian $\hat{\mathbf{J}}^{\dagger}$ includes the weighting matrix $\mathbf{W}$ from equation 4.35. This weighting matrix can be modified to achieve joint limit protection. If it appear that some of the joints are about to approach a limit the weighting matrix from 4.35 is replaced with the matrix

$$\mathbf{W}_{\rho}\,(\boldsymbol{\theta},\,\dot{\mathbf{p}}) = \begin{bmatrix} w_1\,\rho_1(\boldsymbol{\theta},\,\dot{\mathbf{p}}) & 0 & 0 & 0 \\ 0 & w_2\,\rho_2(\boldsymbol{\theta},\,\dot{\mathbf{p}}) & 0 & 0 \\ 0 & 0 & w_3\,\rho_3(\boldsymbol{\theta},\,\dot{\mathbf{p}}) & 0 \\ 0 & 0 & 0 & w_4\,\rho_4(\boldsymbol{\theta},\,\dot{\mathbf{p}}) \end{bmatrix}, \tag{5.4}$$

where $\rho_i(\boldsymbol{\theta}, \dot{\mathbf{p}}),\ i = 1, 2, 3, 4$ are a protection weights that for each joint weights down the joint if it approaches the limit of its working range. The protection weights are calculated with algorithm 1. In algorithm 2 the new velocities for the joints are calculated.

Algorithm 1 Calculate protection weights $\rho(\boldsymbol{\theta},\,\dot{\mathbf{p}})$ to avoid joint limits

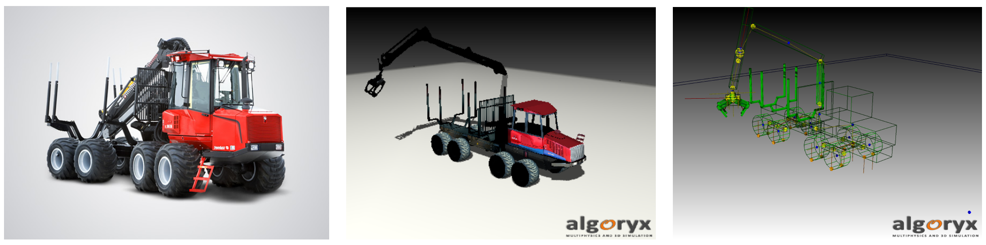
```
Require: $\boldsymbol{\theta}, \dot{\mathbf{p}}$
Ensure: $\rho_i(\boldsymbol{\theta}, \dot{\mathbf{p}}), \quad i = 1, 2, 3, 4$
        $\dot{\boldsymbol{\theta}}_*$
  $\rho_1(\boldsymbol{\theta}, \dot{\mathbf{p}}) = \rho_2(\boldsymbol{\theta}, \dot{\mathbf{p}}) = \rho_3(\boldsymbol{\theta}, \dot{\mathbf{p}}) = \rho_4(\boldsymbol{\theta}, \dot{\mathbf{p}}) = 1$
  $\varepsilon$ = a small value.
  Calculate $\dot{\boldsymbol{\theta}}_*$ from equation 4.37 with the weighting matrix in equation 4.35.
  for $i = 1$ to $i = 4$ do
      $\theta_i^{max}$ = maximum allowed value for $\theta_i$
      $\theta_i^{min}$ = minimum allowed value for $\theta_i$
      $\delta_i < (\theta_i^{max} - \theta_i^{min})\,\varepsilon$
      if $[(\theta_i^{max} - \theta_i < \delta_i)$ and $(\dot{\theta}_{*,i} > 0)]$ or $[(\theta_i - \theta_i^{min} < \delta_i)$ and $(\dot{\theta}_{*,i} < 0)]$ then
          $\rho_i(\boldsymbol{\theta}, \dot{\mathbf{p}}) = 0$
      end if
  end for
```

Algorithm 2 Calculate new joint velocities

```
Require: $\boldsymbol{\theta}, \dot{\mathbf{p}}$
  Calculate $\rho(\boldsymbol{\theta}, \dot{\mathbf{p}})$ and $\dot{\boldsymbol{\theta}}_*$ according to algorithm 1.
  if $\rho_i(\boldsymbol{\theta}, \dot{\mathbf{p}}) \neq 1$ for $i = 1, 2, 3, 4$ then
      $\dot{\boldsymbol{\theta}} = \hat{\mathbf{J}}^{\dagger}\,\dot{\mathbf{p}}$, where $\hat{\mathbf{J}}^{\dagger}$ is calculated with the weighting matrix in equation 5.4.
  else
      $\dot{\boldsymbol{\theta}} = \dot{\boldsymbol{\theta}}_*$
  end if
  Update joint velocities according to $\dot{\boldsymbol{\theta}}$.
```

5.2.3 Modeling of mechanical systems in the virtual environment

In the virtual environment a mechanical systems is constructed with rigid bodies and constraints, where the constraints are used to connect rigid bodies to each other. Rigid bodies and constraints are described in section 4.1. In ATVS there are sensors and



(a) Valmet 830. (b) Virtual environment. (c) Modeled with rigid bodies and constraints.

Figure 5.5: How the *Valmet 830* is modeled in the virtual environment.

actuators implemented for revolute and prismatic constraints. AgX also provide other types of constraints, but these two types of constraints is enough to create complicated mechanical systems.

When modeling a mechanical system typically very rough shapes are used to approximate the shapes of the real system. This approximation is done to simplify calculations in collision detection an contact determination. It is important that the dimensions of the simulated system is identical to the dimensions of the real system, that is the measurements between the joints should coincide in the two systems. It is also important that masses are the same in the two systems. Material parameters can be changed in the physics engine to create realistic behavior for the bodies in the simulation. For example there are parameters to achieve the right friction properties.

When a mechanical system is modeled it is very important that it is in a valid initial state. That is all rigid bodies must be positioned in a way that the real system allows. That is no penetration of rigid bodies, the rigid bodies are not allowed to violate any joint constraints. If the system is in an invalid initial state it will probably "explode" when the simulation is started due to forces that arise from the invalid state.

In figure 5.5 a real Valmet 830 is shown together with the same machine in the virtual environment and an illustration of how the forwarder is modeled with rigid bodies and constraints. In the virtual environment the shape of the forwarder looks just as the shape of the real forwarder, the rigid bodies who approximate the structure of the forwarder are not visible. A rigid body's shape is not visible, it is used for collision detection in the physics engine and does not need to be as detailed as the visible 3D-model. How the actual rigid bodies looks like is illustrated in figure 5.5c. As the figure illustrates the wheels are approximated with cylinders, the links in the crane as well as the machine body are approximated with rectangular boxes, the claws in the grapple are approximated with cylinders etc.

A rigid body can consist of none, one or several collision geometries. The green shapes in figure 5.5c are collision geometries, so every shape in the figure does not correspond to a single rigid body. As stated before the collision geometries are used in collision detection between bodies in the simulation. A rigid body without collision geometries can not collide with other objects, which sometimes can be practical. The grapple for example includes some rigid bodies without collision geometries, these bodies are there for an internal mechanical function in the grapple, but does not need to generate collisions with other objects. In figure 5.5c the constraints are illustrated with a yellow sphere and a yellow line for the constraint axis. There are respectively around 25 rigid

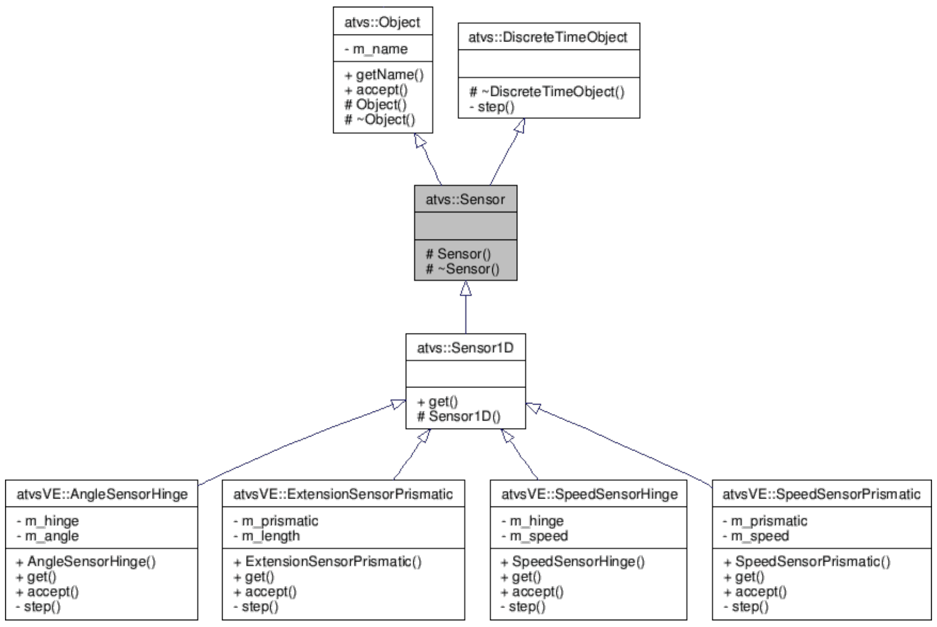


Figure 5.6: Inheritance diagram for the sensor interface.

bodies and constraint in the forwarder model.

5.2.4 Sensor implementation for the virtual environment

Speed sensors and angle/extension sensors are implemented for revolute and prismatic constraints. In the AgX physics engine there are methods supplied for reading speed and angle/extension on these constraints, so on each update the sensor implementation in `ATVS` just read the value from the constraint. In figure 5.6 the inheritance diagram illustrates how the sensor interface in the `atvs` library is implemented for sensors in the virtual environment.

5.2.5 Actuator implementation for the virtual environment

Two types of actuators are implemented for the virtual environment, a hinge motor and a linear motor. The hinge motor is built upon a revolute constraint and the linear motor is built upon a prismatic constraint.

Hinge Motor

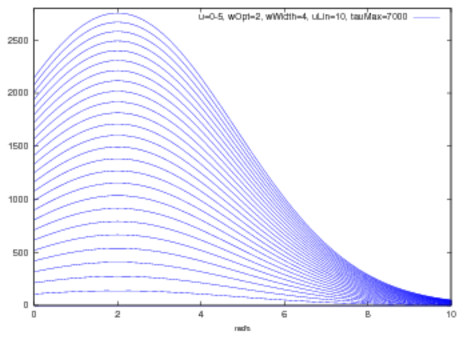
The hinge motor implementation applies a torque on the bodies connected to the hinge (revolute) constraint according to the function

$$f_{hm}(u,\omega) = sign(u)\ \tau_{max}\ e^{-\ \delta(\omega)}\ \ (1 - e^{-\ \beta(u)}\ ), \tag{5.5}$$

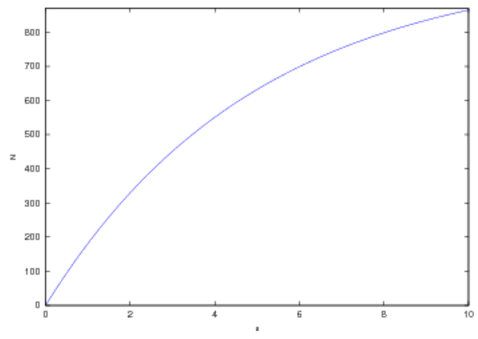
where

$$\begin{aligned} \delta(\omega) &= \tfrac{|(|\omega|-\omega_{opt})|^2}{\omega_{with}^2} \\ \beta(u) &= \tfrac{|u|}{u_{lin}} \end{aligned} .$$

Where u is the motor control signal, ω is the angular velocity in the hinge constraint, u_{lin} is a constant for the slope of the function, ω_{opt} is the optimal angular velocity for the motor, ω_{width} determines the width of the motor function, τ_{max} is the maximal torque that could be generated by the motor and $sign(u)$ is a function that returns -1 for negative values of u and 1 for positive values of u. The torque calculated with the function in equation 5.5 is applied with opposite signs on the two bodies in the constraint according to the direction of the hinge axis. $\beta(u)$ and $\delta(\omega)$ are generally chosen and can be replaced with more specific functions identified from a real systems. For example friction models and delays for an actuator can be determined and implemented. Figure 5.7a shows how the torque varies according to the rotational velocity for some values of u.



(a) Hinge motor function. $u : 0 \rightarrow 5$



(b) Linear motor function. $u_{lin} = 5, f_{max} = 1000$

Figure 5.7

Linear Motor

The linear motor implementation applies a force on the bodies connected to the prismatic constraint according to the function

$$f_{lm}(u) = sign(u)\ f_{max}\ (1 - e^{\,-\beta(u)}\ ) \tag{5.6}$$

where

$$\beta(u) = \frac{|u|}{u_{lin}},$$

along the prismatic axis. Here u is the motor control signal, f_{max} is the maximal force that can be generated by the motor, u_{lin} is a constant for the slope of the function and $sign(u)$ is a function that returns -1 for negative values of u and 1 for positive values of u. The force calculated with the function in equation 5.6 is applied with opposite signs on the two rigid bodies according to the prismatic axis. As for the hinge motor $\beta(u)$ can be replaced with a function identified from a real system. Figure 5.7b shows the force generated by the linear motor function with $u_{lin} = 5$ and $f_{max} = 1000$.

Chapter 6

Proposals on shared control architectures

Control architectures are briefly covered in section 2.2, and as described in section 2.2.2 shared control allows an operator and a control loop on the *slave* side to control the same task. This chapter gives proposals on how shared control could be achieved on different levels with sensors, PID controllers and actuators. For example the control can be shared on a low control signal level or at a higher level where the operator interact with a trajectory tracker or a motion planner.

A motion planner plans a motion for a mechanical system, this can be done as a pre process before a task is started or in real time while the task is executed. The trajectory tracker track the motion planned by the motion planner.

If the joints in a mechanical system are supplied with sensors and actuators, it is then possible that for each joint use a PID control algorithm to control the joint. The actuators can be hydraulic cylinders, DC-motors etc. The sensors can measure angle, extension, translational- and rotational speed etc in the joints. The output from a PID control algorithm depends on a reference value and the current value on the parameter that should be controlled. The reference value is the desired value for the controlled parameter. So if the angle in a joint is controlled with a PID control algorithm and the desired angle is π radians, then the reference value should be set to π radians and the PID control algorithm calculates a control signal to the joint actuator. The control signal, which depends on the reference value and the current angle of the joint, should make the actuator achieve the reference value. PID control is further described in section 4.2.1.

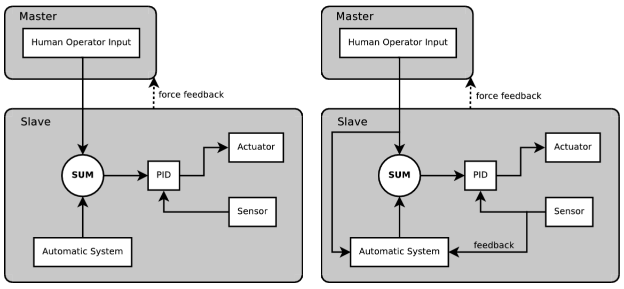
In section 6.1 two shared control architectures are proposed, the operator and the local control loop in the *slave* shares the control of a task by interacting with the reference values to the PID controllers. Section 6.2.1 give proposals on a shared control architecture where the interaction between the operator and the local control loop in the *slave* are done on the motion planning and trajectory tracking level. In the architectures described in both these sections feedback from the *slave* to the operator (*master*) can be achieved through force feedback in the operators controlling devices.

6.1 Interact with reference values.

Shared control can be performed on a low level by interaction with the control signals to a controller or actuator. A control architecture where the interaction are performed at the control signals to the actuator were discarded. It was discarded because of the fact that a PID controller is a feedback controller and will try to compensate for the operators contribution in the actuator control signal. Instead an architecture where the interaction is done at the reference signals to the PID controller was chosen. A PID controller takes a reference value, also referred to as set point, as input and with the help of sensor data it tries to achieve the reference value in the controlled parameter with the help of an actuator. Here it is proposed that the reference signal produced by the local control loop on the *slave* side is simply added to the reference signal from the human operator.

6.1.1 Without control parameter feedback

Figure 6.1a shows the shared control architecture without feedback from the controlled parameter available to the control algorithm in the *slave*. In the figure, *"Automatic System"* corresponds to the control algorithm that generates reference values to the PID controllers. The operator can also contribute with a reference signal for the joint, the operators reference signal and the control algorithm's reference signal are then added together before it reach the PID controller. Without feedback from the controlled parameter the automatic system is an open loop system and is therefore unaware of the actions in the system caused by the operator. An example of this architecture is when the control algorithm calculate a velocity reference for a joint, but it does not need measurements on the joint for the calculation.



(a) Without feedback. (b) With feedback.

Figure 6.1: The shared control architecture where the *master* and the *slave* interact through the reference signal to the PID controllers.

6.1.2 With control parameter feedback

In a shared control architecture with feedback from the controlled parameter, the control algorithm in the *slave* typically use the feedback for calculating the reference values to the PID controllers. The change caused by the human operator can then be perceived as a disturbance by the control algorithm. Therefore the change caused by the human operator can be notified to the control algorithm so it does not perceived the operator action as a disturbance. In figure 6.1b the *"Automatic System"* is the control algorithm in the *slave* that generates reference values to the PID controllers. As illustrated the control algorithm is provided with feedback from the sensor of the controlled parameter and it is also provided with the operators reference signal. A case that illustrates this shared control architecture is when the velocity references calculated for a joint depends on the current velocity or position. The control algorithm are here aware of the operators actions and can then include them in the reference signal calculations.

Gains can also be introduced to the reference signals from both the operator and the control algorithm before they are added together. The gains can be selected according to the operators actions to get smooth transitions between the operator and the control algorithm. This is not illustrated in the figure.

6.2 Interaction on a higher level

In the previous section shared control at the PID controllers reference signals where considered, this section propose shared control on a higher level, the control algorithm level. This shared control architecture is illustrated in figure 6.2. The human operator interact with the high level control algorithm which then regulate the PID controllers.

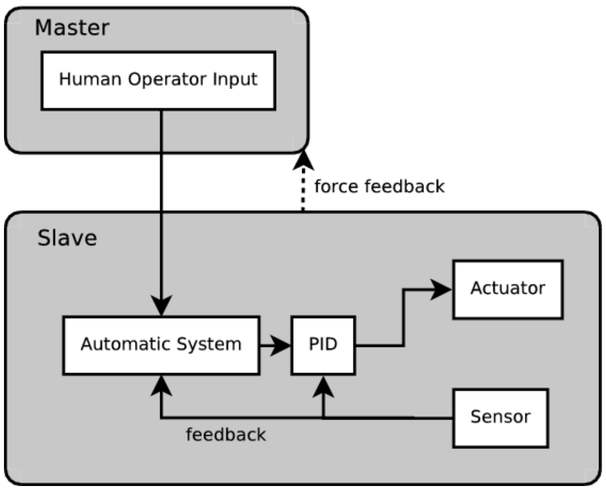


Figure 6.2: A high level shared control architecture.

A working cycle for an autonomous robot manipulator typically includes to follow a trajectory, this trajectory can be pre calculated or calculated by a motion planner online. High level shared control on a trajectory tracking level and the motion planner level are discussed further here. Section 6.2.1 discuss shared control interaction with the trajectory and the trajectory tracker. A shared control tracking algorithm is presented in section 6.2.2. Section 6.2.3 discuss shared control interaction with the motion planner.

6.2.1 Interact with the trajectory and the trajectory tracker

The trajectory tracker's task is to track a defined trajectory. A trajectory can be defined in several ways, for shared control it is important to distinguish between time independent and time dependent trajectories. Time dependent trajectories are defined by positions that should be reached at a certain time while time independent trajectories only depends on the positions along the trajectory. For shared control it is easiest to work with time independent trajectories because then there is no need to recalculate the time dependency when the operator affect the motions.

If the trajectory is time independent and the trajectory tracker tracks the trajectory by velocity references for the tip of the manipulator, further on called end effector, then shared control can be achieved by merging velocity references from the operator and the trajectory tracker. Section 6.2.2 presents an algorithm based on a time independent trajectory and interaction through end effector velocity for a manipulator.

Another approach is that the operator interact with the trajectory by making changes in the parameters that defines the part of the trajectory that is ahead of the end effector. Also in this approach time dependent trajectories are complicated, changing the defining parameters of the trajectory implies recalculation of the time dependency. Therefore only time independent trajectories are considered further on.

6.2.2 A velocity based trajectory tracking algorithm with shared control

This section introduce an algorithm that tracks the end effector of a robot manipulator to a time independent trajectory and allows shared control between the the trajectory tracker and the operator. The algorithm use velocity references and requires an inverse kinematics algorithm for the manipulator to translate an end effector velocity to joint velocities. The algorithm is based on an algorithm called *Follow the carrot* described in [8]. With velocity references the algorithm makes the end effector strives towards a point

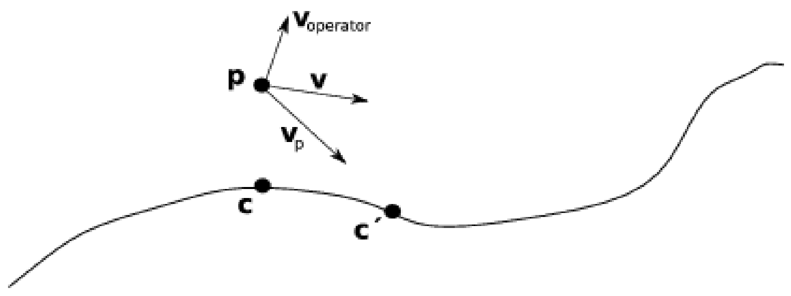


Figure 6.3: Follow the carrot trajectory tracking with shared control.

ahead of it along the trajectory, this point is called the *carrot point*, thereof the name *Follow the carrot*. Figure 6.3 illustrates the algorithm. From the current end effector position $\mathbf{p}$, the closest point $\mathbf{c}$ on the trajectory is calculated. The end effector deviation from the trajectory is used to calculate a *look ahead distance*, which is a measure on how far ahead along the trajectory from the closest point, $\mathbf{c}$, the carrot point $\mathbf{c}'$ should be located. The carrot point is the point that the algorithm aims the end effector velocity at. The *look ahead distance* increase with increasing end effector deviation and decrease with decreasing end effector deviation. So the distance between the carrot point, $\mathbf{c}'$, and the closest point, $\mathbf{c}$, is a function of the end effector deviation. The final end effector velocity $\mathbf{v}$ is calculated as a sum, weighted sum or a mean value of the commanded velocity $\mathbf{v}_{operator}$ from the operator and the velocity $\mathbf{v}_p$ calculated from the current end effector position and the carrot point. The operators velocity reference, $\mathbf{v}_{operator}$, will typically cause the end effector to deviate from the trajectory and make the *look ahead distance* to increase. If the operator stops to contribute with velocity references the end effector will smoothly return to the trajectory and the *look ahead distance* decrease. The algorithm is described in algorithm 3.

Algorithm 3 Follow the carrot trajectory tracking with shared control

```
Require: A trajectory defined as line segments. Magnitude $k$ of the tracking velocity
    and the velocity contribution $\mathbf{v}_{operator}$ commanded by the operator.
Ensure: An end effector velocity that makes the end effector follow the trajectory, the
    end effector velocity can be affected by the operator.
  1. Determine the current position $\mathbf{p}$ of the end effector.
  2. Find the point $\mathbf{c}$ on the trajectory that is closest to the end effector position $\mathbf{p}$.
  3. Calculate the look ahead distance proportional to the end effectors deviation, $||\mathbf{c} - \mathbf{p}||$, from the trajectory. An increased deviation correspond to an increased look ahead distance.
  4. Find the carrot point $\mathbf{c}'$. Which is at a look ahead distance from the closest point $\mathbf{c}$ along the trajectory.
  5. From the end effector position and the carrot point, calculate the velocity vector $\mathbf{v}_p = \frac{\mathbf{c}'-\mathbf{p}}{||\mathbf{c}'-\mathbf{p}||} k$.
  6. Calculate the final velocity vector $\mathbf{v} = \mathbf{v}_p + \mathbf{v}_{operator}$
  7. Calculate joint velocities from the final end effector velocity $\mathbf{v}$.
```

6.2.3 Interact with the motion planner

The motion planner generates a trajectory that the trajectory tracker should follow. If the trajectory can be generated at the same time as the tracker tracks the trajectory or if the generation of a complete trajectory is fast enough to be generated in one simulation step, it is called *online* generation. If the complete trajectory is generated as a pre process independent of the simulation steps, it is called *offline* generation. In both cases time independent trajectories defined with as few points as possible, without modifying the original shape of the trajectory, is faster to generate than time dependent trajectories which typically includes a point for every time step of the tracking task.

The operator can interact with the motion planner instead of interacting with the trajectory or the trajectory tracker. Typically the motion planner have information about obstacles and from that it plans a trajectory from the current position of the end effector to a goal position.

For the case when the trajectory is generated online the operators input signals can be used to introduce a way point between the current end effector position and the goal position that the trajectory must pass through. The motion is then planned from the start point to the way point, from the way point to the end point.

With offline trajectory generation the trajectory have to be regenerated during several time steps when there is input from the operator. So its very important that the trajectory can be generated fast enough so the end effector motion is minimal during the generation. Time dependent trajectories should therefore be avoided in this case in favor for simpler time independent trajectories.

6.3 Force feedback

In figures 6.1 and 6.2 force feedback is illustrated as a dotted arrow from the *slave* to the *master*. This is an abstract illustration showing that there can be force feedback from the *slave* to the *master*, but it does not say anything of how this is done. That is because force feedback does not have to be included in the control algorithm.

Force feedback can depend on sensor measurements that is not included in the control algorithm. For example one could imagine that sensors measuring the pressure of the oil current in a hydraulic system can be used to determine the load in joints and with force feedback illustrates for the operator which motions that requires a higher respectively lower pressure of the oil current. This can cause many combinations of systems that includes different types of force feedback, therefore force feedback is just abstractly illustrated in the figures.

Force feedback could also be used to guide the operator in its movements. For example forces in the operators control devices that comport with the movements of following a trajectory can be generated. Force feedback can also be used to avoid that the operator runs the system into an obstacle or notify when joint limits are reached.

Chapter 7

Test results

This chapter demonstrates how `ATVS` can be used to simulate semi-autonomous mechanical systems with shared control in the virtual environment. Two mechanical systems is considered, a simple loading system that operates in two dimensions and a *Valmet 830* forwarder. Some of the shared control architectures proposed in chapter 6 are implemented, tested and very briefly evaluated.

7.1 Loading system

The loading system operates in two dimensions and is equipped with a rotatable grapple, the system is illustrated in figure 7.1. Systems of this type can be used to unload or load cargo from one vehicle to another, move cargo from the vehicle to a cargo store etc. The system in figure 7.1 moves logs from one store to another. Typically algorithms are easier to implement for two dimensions than for three dimensions, so a simple two dimensional system like this makes it possible to easy implement and test shared control approaches.

The grapple can be regulated in height and moved between the two stores, this is done with prismatic joints. The grapple can also be rotated, which is done with a hinge joint. The grapple and the mechanism for the grapple claws are modeled with one prismatic joint and several hinge joints in the same way as a real grapple. There are sensors for measuring the angle or extension on the joints in the system. Linear motors are connected to the prismatic joints that moves the grapple and to the prismatic joint that actuates the claws in the grapple. A hinge motor controls the rotation of the grapple. PID controllers are tuned for velocity control of the joints, so to control a joint the operator gives a velocity reference to the PID controller of that joint. The operator use two joysticks of the same type as in figure 5.4 to control the velocity references.

7.1.1 Shared control tests

Chapter 6 propose some shared control architectures for systems with PID controllers, actuators and sensors. Some of these are implemented for this loading system and are briefly described and evaluated in this section.

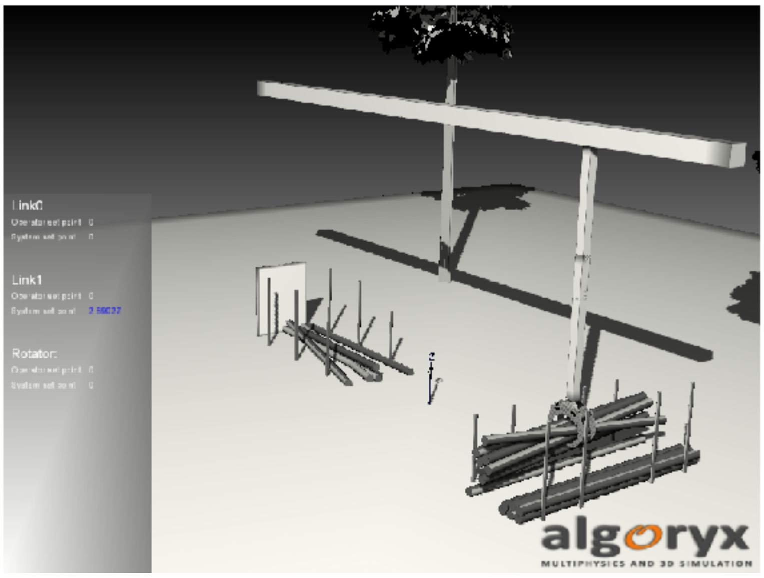


Figure 7.1: The loading system developed to test shared control.

Reference interaction without feedback

Section 6.1.1 describes a shared control architecture where reference values from the operator and the *slave's* local control algorithm are merged together. Further the described architecture lacks grapple position feedback for the local *slave* control algorithm. A variant of that architecture are here tested with the loading system.

An automated task takes the grapple from one of the log–stores to the other while the grapple is rotated to fit in the target log–store. The automated task is started by pushing a button on one of the joysticks, the automated task can be interrupted by pushing another button on one of the joysticks. There are three different states that the automated task can take: *go up from current store*, *move to the target store* and *go down in the target store*. The automated task gives velocity references to the joints depending on the state of the task and the state of the task is changed when the grapple pass through predefined way points. For example, when the grapple is located above a certain height limit at the same time as it is above the starting log–store, the state is *move to the target store*. When the grapple then reaches the target log–store the state is changed to *go down in the target store*. When the grapple approaches the target position the task smoothly slows down the velocity.

During execution of the task the operator can then contribute to the velocity reference values, practically the human operator contribute to the velocity references by moving the joysticks in the same way as in manual control. The fact that the control algorithm does not have any feedback about the current velocity at the joints will make it to blindly give velocity references according to the state of the task. The velocity contribution from the operator will not affect the control algorithm. To get smooth interaction between the operator and the control algorithm gains on their respectively velocity reference can be introduced. These gains can depend on the magnitude of the operators joystick movements.

Figure 7.1 shows the loading system in the beginning of the task, the state of the task is *go up from current store*. To the left in figure 7.1 the velocity reference for each joint, from both the operator and the control algorithm, are illustrated. In the figure the control algorithm moves the grapple upwards, which is the result of the velocity reference shown in blue for Link 1. The operator does not give any velocity references in this figure.

With the automated tasks that moves the grapple from one log-store to the other, the operator gets micro–pauses in the workload and the level of mental stress may be decreased. If the operator need to adjust the movements of the automated task, the task would not notice that. For example, if the logs are about to collide with the poles that surrounds each log–store the operator can slow down the movements of the grapple and if it is needed raise the grapple to rotate the logs to fit into the log–store. Then the operator let the task move down the grapple into the log–store while the operator smoothly take over the control to do manual unload.

The interaction between the operator and the control algorithm feels very natural and the transitions between manual control and autonomous control at the beginning and end of the task goes smoothly due to the shared control implementation.

Reference interaction with trajectory tracking

A shared control architecture where interaction occurs at the level of PID controller reference values that also includes trajectory tracking can be viewed as a hybrid among the architectures described in sections 6.1.1, 6.1.2 and 6.2.1. The same loading system as described previously is still used but with a different control architecture. As previously

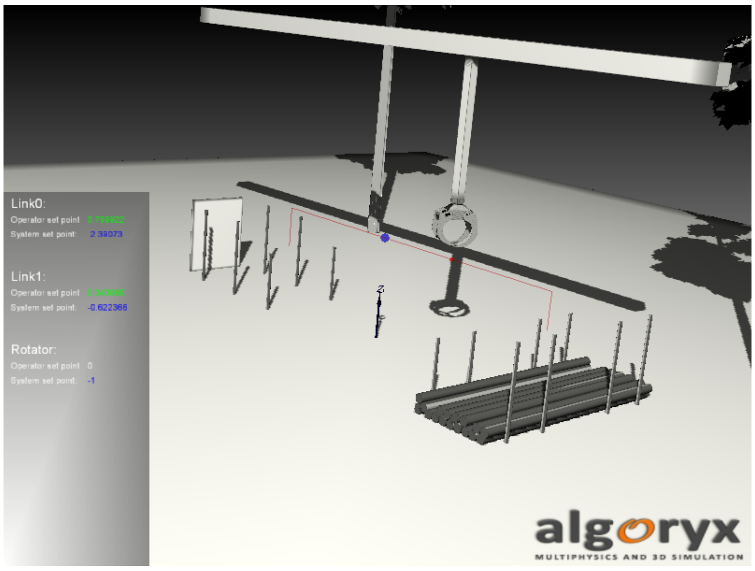

Link0:
Operator set point 0.799922
System set point: 2.39073
Link1:
Operator set point 0.343969
System set point: -0.622366
Rotator:
Operator set point 0
System set point: -1
z
algoryx
MULTIPHYSICS AND 3D SIMULATION

Figure 7.2: Trajectory tracking with shared control.

a task moves the grapple from the current log–store to the other log–store. But the difference is that now a trajectory is generated and tracked by a velocity based trajectory tracker. The trajectory tracker use algorithm 3 described in section 6.2.2, but the algorithm is slightly modified to fit the case where the interaction between the operator and the control algorithm are at the reference level. The difference from algorithm 3 is that the velocity $\mathbf{v}_{operator}$ from the operator is excluded from the algorithm and the velocity $\mathbf{v}_p$ is the reference value from the control algorithms to the PID controller.

The interaction with the operator is then at a velocity reference level before the PID controllers, where the velocity references from the operator and the control algorithm is merged together. Inputs from the operator will result in a grapple deviation from the trajectory, the control algorithm then strives to return the grapple to the trajectory. To tune the algorithm for smoother interaction between the operator and the control algorithm, gains on the velocity references that depends on either the end effector deviation or the operators velocity reference can be considered. The control algorithm do not use the joint velocity as feedback term, instead the feedback to the control algorithm is the position of the grapple, which determine the deviation from the trajectory. The trajectory tracker smoothly accelerate the tracking at the beginning of the trajectory and smoothly slows down the tracking at the end of the trajectory.

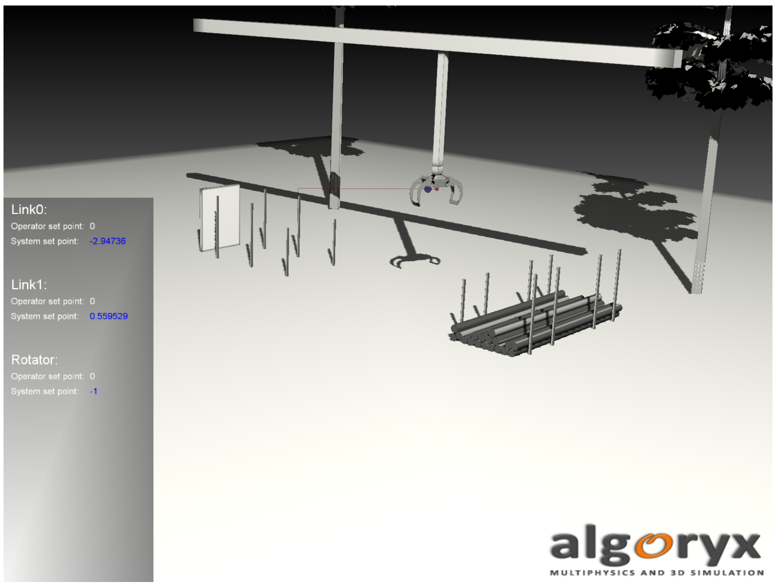
Figure 7.2 shows the loading system with shared control at the velocity reference level and trajectory tracking. In the figure the trajectory is denoted with a red line, the small red sphere on the trajectory is the point on the trajectory that is closest to the grapple and the larger blue sphere is the carrot point. The carrot point is described in section 6.2.2. The control algorithm strives the grapple towards the red sphere, the distance between the red and the blue sphere on the trajectory increase with increased deviation of the grapple. That is to get a smooth return to the trajectory when the operators velocity reference decrease. To the left in figure 7.2 contributions to the reference values from both the operator and the control algorithm is shown. It can be seen that both the operator and the control algorithm contributes with reference values to the joints.

As previously mentioned the operators mental strain may decrease when autonomous tasks introduce small pauses in the operators work cycle. The above described interaction technique gives a smooth and natural interaction between the operator and the autonomous task. A drawback with this method is that the visual indications of the trajectory are very important for the interaction. With lack of the visual indicators the interaction feedback is lost and the interaction does not become as intuitive as before. Without visual feedback the feedback have to be replaced with some other feedback, here force feedback in the operators joysticks are a natural solution. But this is not explored in this thesis and left for future work.

Regeneration of the trajectory

In section 6.2.3 a high level shared control architecture that interact with the motion planner is described. This architecture is explored with the loading system. As before a trajectory is planned and generated. The trajectory is tracked in the same way as in section "*Reference interaction with trajectory tracking*". But now the operators input generates new trajectories. For example when the operator moves the joystick a start point is predicted from the current position of the grapple and the input from the operator. A new trajectory is then generated from the predicted starting point and the tracker then tracks the new trajectory.

This is illustrated in figure 7.3 where it can be seen that the trajectory is changed compared to the original trajectory. The original trajectory is the same as the trajectory in figure 7.2. The interaction between the operator and the control algorithm is at least as good as for the previous cases, except for when the operators input coincide with the direction of the trajectory. The trajectory is tracked with a velocity based algorithm, when the operators input creates new starting points that lies on the trajectory, this



Link0:
Operator set point: 0
System set point: -2.94736

Link1:
Operator set point: 0
System set point: 0.559529

Rotator:
Operator set point: 0
System set point: -1

Figure 7.3: Motion planner interaction.

means that the new trajectory is the same as the old trajectory. The operator must create new trajectories that deviates from the current trajectory, also start points for new trajectory's that are created behind the grapple will have no effect, because the tracking algorithm never look backwards. Motion planning interaction is also dependent on the motion planning algorithm and the complexity of the algorithm, see section 6.2.3.

Motion guidance with force feedback

For the loading system force feedback in the joysticks to guide the operator is explored. Way points from the start position to the end position form the trajectory that should be followed. A force depending on the distance and direction to the next way point is generated in the operators joysticks. The magnitude of the force is larger with a larger distance to the way point. The method gives a good guidance for the operator but have not been deeply tested because of time limitations in the project.

7.2 Valmet 830 forwarder

A forwarder is used for transportation of logs from the felling place to a roadside, where the logs are loaded on a lumber car for further transportation. The forwarder is equipped with a redundant crane , see section 4.3 for more information on redundant manipulators and how velocity control of the boom-tip can be implemented. The forwarder crane operates in 3 dimensions and is therefore more complicated than the loading system.How the complete forwarder is modeled in the virtual environment is briefly covered as an example in section 5.2.3.

There are sensors available for reading angle/extension and velocity for all joints in the crane and the grapple. For the machine the same sensors are available for every wheel joint and for the steering link. There are also motors to all above mentioned joints.

For the forwarder crane there are some simplifications. The joints θ_2 and θ_3, which are described in section 4.3, are modeled as hinge motors. In reality they are actuated by linear motors in the form of hydraulic cylinders that are attached to the links connected with the joint. But the fact that the angle and velocity is measured in the joint and that the motor is controlled by a PID controller makes the type of the underlying actuator of less importance. One can consider that the control algorithm will work identical if a well tuned PID controller is supplied for each joint regardless of the actuator type and given that angle and velocity is measured in the joint.

There are PID controllers tuned for velocity control of each joint in the forwarder crane and its grapple. Further there are tuned controllers for velocity control of the wheels and the steering joint. In the virtual environment the full forwarder is controlled with two joysticks of the same type as in figure 5.4.

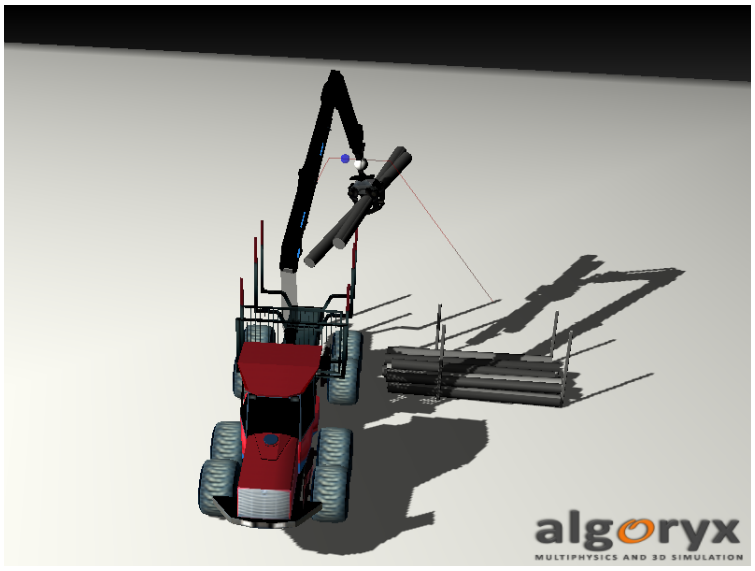


Figure 7.4: Automated loading task.

7.2.1 Determination of tasks that could be automated

To get an insight in how the operator operates the crane, movies with professional forwarder operators have been studied. From the movies conclusions can be drawn on which parts of the working cycle that can be automated, where in the working cycle the operator performs very complicated tasks, how fast the operators working cycle is, etc.

If the automated tasks should be lucrative, the automated working cycle should be at least as fast and smooth as the non-automated working cycle performed by a professional operator. If this is fulfilled it will be easier for an unexperienced driver to become lucrative. For the professional driver the total time of a working cycle may be significantly reduced while the short pauses in the working cycle introduced by the automated tasks may reduce the operators mental strain [11, 10].

The movies shows that the operator performs complicated tasks when grabbing and releasing the logs, while the transport between ground and load bunk is quite simple. It is also obvious that the grabbing of logs is much more complicated than releasing logs. By automating the simple part of the working cycle the operator can get short pauses with possibilities to relax between the more complicated parts of the working cycle.

From this information and previous experience in the area the transportation parts of the working cycle is chosen to be automated. These are the parts when the grapple goes from the bunk and out to grab a log and after a log is manually grabbed the grapple goes back to the bunk. Grabbing and releasing of logs are chosen to be manually controlled. Figure 7.4 shows the autonomous loading task.

7.2.2 Shared control test

For the forwarder a high level shared control architecture described in section 6.2 is tested and evaluated. A trajectory is generated from the current position of the grapple to a target point. This is done with a very simple motion planner that during planning of trajectories only is aware of the positions of the loading bunk and the rest of the machine. The trajectory is tracked with the velocity based tracking algorithm described in section 6.2.2. This section also describes how the operator interact with the control algorithm described in algorithm 3 on page 43.

The grapple is attached to the forwarder crane without any control possibilities except for the rotator. That is, the grapple freely swings around two hinge joints in the boom tip of the crane. There are dampers, but there is no ability to control these two joints. This makes it a little bit more complicated to control the crane, if for example a log is not grabbed at the middle it will not be horizontal during the transportation to the loading bunk. Also if there are heavy movements in the crane they may cause large swinging in the grapple.

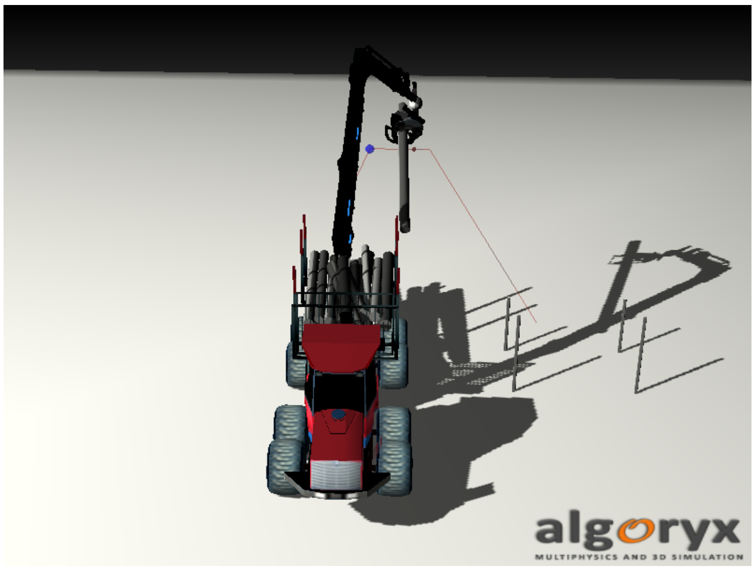


Figure 7.5: The loading task with shared control illustrated.

Figure 7.4 shows the execution of an automated loading task. In this figure the logs are grabbed at the middle and the grapple lies horizontal during the task, so the operator does not need to take action during the task. The operator instead smoothly takes over the control at the end of the task and release the logs in the loading–bunk.

In figure 7.5 a log is grabbed at the very end. This introduce swinging in the grapple when the crane is moved and the grapple will be horizontally aligned. As illustrated in the figure the operator have to take action an brake the tracking of the trajectory and at the same time raise the grapple so the log does not collide with the machine.

The interaction between the operator and the control algorithm is natural and smooth as for the same case on the previously described loading system. As with the loading system the intuitivity in interaction will be heavily decreased without any visual feedback. Therefore force feedback in the operators joysticks may be investigated.

Chapter 8

Conclusions

In this master thesis the framework ATVS is developed. The framework provide functionality to control mechanical systems with PID control. A virtual environment where mechanical systems can be simulated and controlled is supplied with ATVS. The virtual environment is based on the Physics engine AgX from Algoryx Simulations [1] and the rendering toolkit OpenSceneGraph [2].

The virtual environment makes it easy to create models of mechanical systems and evaluate control algorithms together with human interfaces to these mechanical systems. Proposals on automated tasks and shared control have been given for two mechanical systems, a loading system, see figure 7.1, and a Valmet 830 forwarder, see figure 1.2. Inverse kinematics for the forwarder crane, which is a redundant manipulator, have successfully been implemented and used in the automated tasks for the forwarder. Support for force feedback devices is also provided. An application that demonstrates an example of shared control with force feedback are supplied as well as other applications that demonstrates automated tasks with shared control.

The purpose and objectives in chapter 3 are with the above summary of this master thesis considered to be fulfilled.

8.1 Restrictions and limitations

The force feedback system based on gaming force feedback devices and Microsoft's DirectInput API does currently not meat the real time requirements. The constant forces for a force feedback device can only be updated approximately 5 times per second.

8.2 Future work

In future work the ATVS framework could be used to developed new control algorithms and investigate human machine interfaces. Human machine interfaces that are of interest to investigate are force feedback as a replacement for visual feedback indicators and force feedback to guide the operator in its movements of the control devices.

Another interesting area is how motion planning algorithms should be designed and developed to fit with shared control architectures.

For the ATVS framework better support for input- and haptic devices is desirable. Extensions with possibilities to connect real sensors and actuators to ATVS together with an interface to simulink should significantly increase the possibilities of develop, investigate and evaluate control algorithms and shared control. Simulink is a MathWorks software for modeling and simulating dynamic systems based on the Matlab environment [23].

Chapter 9

Acknowledgements

First of all I want to pay attention to my external supervisor Martin Servin which has contributed with his knowledge and spent a lot of time in discussions with me during this thesis. I also want to direct many thanks to Anders Backman and Mattias Linde at Algoryx Simulations for support with AgX. Finally I want to thank my office colleagues Joakim Lundin, Jens Zamanian and Mats Forsberg at the Department of Physics at Umeå University for a great time.

References

[1] AgX Multiphysics Toolkit. `http://www.algoryx.se` (visited February 2009).

[2] OpenSceneGraph. `http://www.openscenegraph.org` (visited February 2009).

[3] World of warcraft. `http://www.worldofwarcraft.com` (visited February 2009).

[4] Oryx Simulations. `http://www.oryx.se` (visited February 2009).

[5] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer, 2008.

[6] Alan Ettlin. *Rigid body dynamics simulation for robot motion planning*. PhD thesis, Lausanne, 2006.

[7] K. J. Åström and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2008. Available online at ”http://press.princeton.edu/titles/8701.html”.

[8] Ola Ringdahl. *Techniques and Algorithms for Autonomous Vehicles in Forest Environment*. Licentiate thesis, Department of Computing Science, Umeå University, 2007. ISBN 978-91-7264-373-4.

[9] Mikael Niva, Andreas Mettävainio, Mia Petersson, and Jill Jonsson. Automatisering av kranrörelse.

[10] Björn Löfgren. Kinematic control of redundant knuckle booms, 2004.

[11] M. Brander, D. Eriksson, and Björn Löfgren. Automation of knuckleboom work can increase productivity. *Skogsforsk Results*, No 4, 2004.

[12] Anton Shiriaev, Leonid Freidovich, Ian Manchester, Uwe Mettin, Pedro La Hera, and Simon Westerberg. Status of smart crane lab project: Modeling and control for a forwarder crane.

[13] Ifor. `http://www.cs.umu.se/research/ifor/` (visited February 2009).

[14] Umeå University. `http://www.umu.se/` (visited February 2009).

[15] Komatsu forest. `http://www.komatsuforest.com/` (visited February 2009).

[16] Anders Backman. Lecture notes from the course ”advanced computer graphics and applications” at [14].

[17] Simon Westerberg, Ian R. Manchester, Uwe Mettin, Pedro La Hera, and Anton S. Shiriaev. Virtual environment teleoperation of a hydraulic forestry crane. In *ICRA*, pages 4049–4054. IEEE, 2008.

[18] Paul G. Griffiths and R. Brent Gillespie. Shared control between human and machine: Haptic display of automation during manual control of vehicle heading. In *HAPTICS*, pages 358–366. IEEE Computer Society, 2004.

[19] P. D. Lawrence, S. E. Salcudean, N. Sepehri, D. Chan, S. Bachmann, N. Parker, M. Zhu, and R. Frenette. Coordinated and force-feedback control of hydraulic excavators. In *The 4th International Symposium on Experimental Robotics IV*, pages 181–194, London, UK, 1997. Springer-Verlag.

[20] Simon P. Levine, David A. Bell, Lincoln A. Jaros, Richard C. Simpson, Yoram Koren, Senior Member, Johann Borenstein, and Navigation System. The navchair assistive wheelchair navigation system. *IEEE Transactions on Rehabilitation Engineering*, 7:443–451, 1999.

[21] Richard Simpson, Simon P. Levine, David A. Bell, Yoram Koren, Johann Borenstein, and Lincoln A. Jaros. The navchair assistive navigation system. In *In IJCAI Workshop on developing AI applications for the disabled*, pages 167–178, 1995.

[22] P. Lim, J. Yang, N. Hildreth, and W. Friedrich. Application of shared telerobotic control for contour following processes, 2002.

[23] Simulink – simulation and model–based design. `http://www.mathworks.com/products/simulink/` (visited February 2009).

[24] Kenny Erleben, Jon Sporring, Knud Henriksen, and Kenrik Dohlman. *Physics-based Animation (Graphics Series)*. Charles River Media, Inc., Rockland, MA, USA, 2005.

[25] Umfpack. `http://www.cise.ufl.edu/research/sparse/umfpack/` (visited February 2009).

[26] Kenny Erleben. *Stable, Robust, and Versatile Multibody Dynamics Animation*. PhD thesis, Department of Computer Science, University of Copenhagen (DIKU), Universitetsparken 1, DK-2100 Copenhagen, Denmark, March 2005. `ftp://ftp.diku.dk/diku/image/publications/erleben.050401.pdf`.

[27] Kenneth Bodin, Martin Servin, Krister Wiklund, and Claude Lacoursière. Lecture notes from the course ”visual interactive simulation” at [14].

[28] B. Thomas. *Modern Reglerteknik*. LIBER, 2003.

[29] Karl Johan Åström. Control system design lecture notes for me 155a. Department of Mechanical and Environmental Engineering University of California Santa Barbara, 2002.

[30] M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, Inc., 2006.

[31] J. J. Craig. *Introduction to Robotics: Mechanics and Control (3 edition)*. Prentice Hall, August 6, 2004.

[32] Samuel R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. typeset manuscript, available from http://math.ucsd.edu/ sbuss/researchweb. *IEEE Journal of Robotics and Automation*, 3:681–685, 2004.

[33] *CMake*. `http://www.cmake.org/` (visited February 2009).