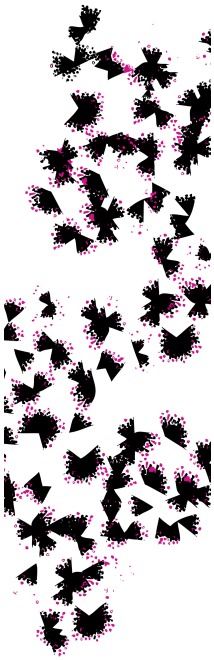

Upgrading the Safety Layer and Demo of the youBot Robot

S.D. (Stefan) Frijnts

Individual Assignment

Committee:
Dr.ir. J.F. Broenink
D. Dresscher, MSc

Juli 2014

Report nr. 007RAM2014
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summery

The youBot is a robotic arm on an omni-directional platform. The youBot is developed for research in control and software development. The youBot of the university of Twente has demo software which is developed using a specially designed toolchain. The toolchain is developed for the European BRICS project. The demo software uses a haptic device for controlling the robotic arm and platform over a wireless connection.

During the project the demo software is updated. In the old demo, the arm and base were modeled as a kinematic link and were controlled using the haptic device. This gave limited control of the platform. Therefore the demo is upgraded in the way the platform and robtic arm can be controlled separately. The base is controlled using a joystick, for the arm a new haptic device is implemented.

The project uses the special designed toolchain for upgrading the demo software. The toolchain consist of Open Robot Control Software (OROCOS), Robot Operating System (ROS) and 20-sim. 20-sim is used for modeling and simulating the control algorithm. The code generation of 20-sim is used for creating the OROCOS software packages which can be used on the youBot for doing the real-time control. ROS is used for the network communication and debugging tools. The toolchain was badly documented. It took some time to get the toolchain working properly.

The youBot demo software uses the passivity layer to ensure stability of the controller. The passivity layer uses an energy buffer which is used for doing the control actions. The control action is decreased when there is not enough energy available in the buffer. During the project the energy buffer was incorrectly filled which allowed the youBot to damage itself. Therefore extra safety layers are added which work in a more robust way and should protect the arm from damaging itself.

On the remote computer runs the setpoint generation software. It uses two joysticks to generate setpoints for the base and arm. A normal joystick is used for controlling the base. The joystick generates a setpoint which is send over the network to the youBot. The youBot has a control loop which sends the youBot base from his actual position to the virtual setpoint. An extra feedback is implemented which prevents the virtual setpoint from moving further when the connection is lost.

The robotic arm of the youBot is controlled using the haptic device. The haptic feedback gives a feeling how the setpoint can be moved taking into account the topological limitation of the robotic arm. This prevents the setpoint from going to a position which is not reachable for the arm. The haptic feedback has no feedback in the rotational direction. This is problematic as the arm is topological limited in rotational directions. To solve this the setpoint is automatically rotated depending on the input of the haptic device. This makes the setpoint rotational reachable for the arm.

The toolchain is successfully used for updating the demo software. Extra documentation is added on the wiki of RaM to make the toolchain better usable.

Samenvatting

De youBot is een omni directional platform met een 5 assige robotarm er op. De youBot is ontwikkeld voor het doen van onderzoek naar software ontwikkeling en control algoritmes. De youBot van universiteit Twente heeft demo software die ontwikkeld is met behulp van een speciaal ontwikkelde toolchain voor het Europese BRICS project. De demo maakt gebruik van een haptiche joystick om de robot arm aan te sturen over een draadloos wifi netwerk.

Het doel van het project is het aanpassen van de huidige demo software. In de oude demo werd de arm en het platform aangestuurd met behulp van één haptiche joystick. Dit gaf een beperkte controle over het platform. Daarom is het platform en robot arm gesplitst naar een normale joystick voor de aansturing van het platform en een haptiche joystick voor de robot arm.

De toolchain integreert Open Robot Control Software (OROCOS), Robot Operating System (ROS) en 20-sim. 20-sim wordt gebruikt voor het modelleren en simuleren van het control algorithm. De Code generatie vanuit 20-sim maakt het mogelijk rechtstreeks OROCOS componenten te genereren die gebruikt worden voor de real-time aansturing van de youBot. ROS wordt gebruikt voor netwerk communicatie en debug tools. De toolchain was slecht gedocumenteerd. Tijdens het project is dit aangepakt zodat deze beter bruikbaar wordt.

De youBot maakt gebruik van een passivity layer om te garanderen dat de controller stabiel blijft. The passivity layer werkt door middel van een energie buffer die verbruikt wordt bij het aansturen van de robotische arm. De arm wordt gestopt als de energie buffer leeg is. Tijdens het project werd het duidelijk dat de passivity layer niet genoeg is om de youBot arm te beschermen tegen beschadigingen. Gedurende het project bleek de buffer niet correct gevuld te worden. Hierdoor kon het dat de youBot zichzelf beschadigde. Daarom zijn er twee extra veiligheids lagen toegevoegd die de arm altijd beschermen.

Op de bedienings computer draait setpoint generatie software. Deze maakt gebruik van de twee joysticks om de youBot aan te sturen. De normale joystick wordt gebruikt voor het aansturen van de het platform. Deze joystick genereert een setpoint die over het netwerk naar de youBot gestuurd wordt. De youBot heeft een control loop om van zijn werkelijke positie naar het setpoint te gaan. Een extra terugkoppeling is toegevoegd die voorkomt dat het setpoint verder gaat als het netwerk wegvalt.

De robot arm wordt aangestuurd met behulp van haptische joystick. De terug koppeling zorgt er voor dat de gebruiker voelt wat de robot arm doet en kan doen. Dit voorkomt dat de gebruikers iets doet wat de youBot helemaal niet kan doen. Het haptische apparaat heeft geen terugkoppeling op rotaties. De arm van de youBot is juist gelimiteerd in zijn rotaties. Om dit op te lossen wordt het setpoint afhankelijk van de haptische invoer automatisch zo gedraaid dat de robot arm het setpoint kan bereiken. Dit voorkomt dat door gebrek van haptische terugkoppeling in de rotaties het setpoint verkeerd wordt aangestuurd.

De toolchain is succesvol gebruikt om de input van de youBot demo aan te passen. Extra documentatie is gemaakt zodat het makkelijker is om aan de slag te gaan met de youBot en toolchain.

Contents

1 Introduction

1.1 Context

Over the years, robotics are applied in more and more situations and applications. Development of new applications take significantly more development time as the complexity grows. The Best Practice In Robotics (BRICS) project was started with the objective to "significantly" speed up the development process of robotic applications. To achieve this, the BRICS project adds more structure to the development process, using development tools, computational models, and functional libraries Bischoff et al. (2010). Within the BRICS project the youBot is used for testing software and control algorithms. The youBot is a mobile manipulator, as shown in Figure 1.1, which is designed by researchers from KUKA.

Figure 1.1: The youBot

Currently, the youBot has demo software which is used for demonstrations. The demo software uses a haptic device to control the youBot arm and base. The demo was first shown on the Automatica 2012 trade fair in Munich Germany. Currently, it is used for giving demos for visitors on open days of the university Twente. The demo software was created for the BRICS project as an use case for a software development process and control

The software development process, which is used for developing the demo software, is based on a toolchain. The toolchain consists of 20-sim, Open Robot Control Software (OROCOS) and Robot Operating System (ROS). The integration between these tools is done in the BRICS project. The toolchain enables the use of simulation and code generation to streamline and speed up the development process of robotic software.

1.2 Problem statement

The youBot demo has two limitations. The first one is the small feedback force generation of the haptic device. During operation changes in the haptic feedback are difficult to feel. The second problem was that the base was difficult to control, the base and arm are controlled as one kinematic link using the haptic device for setpoint generation. There is no trajectory

planning, such that it is not ensured that the base moves to the correct position where the arm can reach the virtual setpoint.

1.3 Goals

The aim of the project is to upgrade the existing demo to give better control and feedback to the user. As described in the problem statement the control of the base is limited. Therefore the control of the base and arm of the youBot will be split up into the arm using the haptic device and the base using the joystick. This will give more control to the base and makes the haptic feedback of the robot arm more consistent. The second improvement is integrating a new haptic device, the Omega 6. The Omega 6 is more accurate and can create a larger force for haptic feedback compared to the old device.

Upgrading the demo requires getting an overview of the existing demo software and toolchain. Limited documentation is available. The first the toolchain needs to be made working and use it to make small changes in the existing demo software. The information on how to use the toolchain Wiki (2014a) and the demo software structure will be documented to the RaM wiki Wiki (2014b). The demo upgrade can be used as example for creating new applications with the youBot.

1.4 Report structure

The report outline is as follows:

- Chapter 2 background information on the youBot and the used toolchain is presented.
- Chapter 3 uses the background information for describing the software upgrades. The safety layer, base and arm arm control are treated in separated sub chapters.
- Chapter 4 the conclusions and recommendations are presented.

2 Background

The youBot is a mechatronic mobile platform. The background information is split up in the youBot hardware mechanics, electrical and software design of the youBot demo software.

The demo software is generated using tools. The tooling integration makes it possible to directly implement the generated code into the real youBot. Information on the used tools can be found in 2.3. An overview of the integration between the tooling is done in 2.4.

2.1 youBot Hardware

2.1.1 youBot mechanics

The youBot is a robotic arm on an omni-directional platform. The arm has five joints, each joint has a motor with a build-in gearbox, relative encoder and joint bearings. All the joints have a mechanical end stop. The gripper has two fingers which can open and close. The gripper finger position is not measured by encoders. During the initialization procedure the gripper fingers are fully opened and closed, to make sure that they are in the closed position after initialization. See Figure 2.1 for the joints and the reference frame of the gripper and the joint.

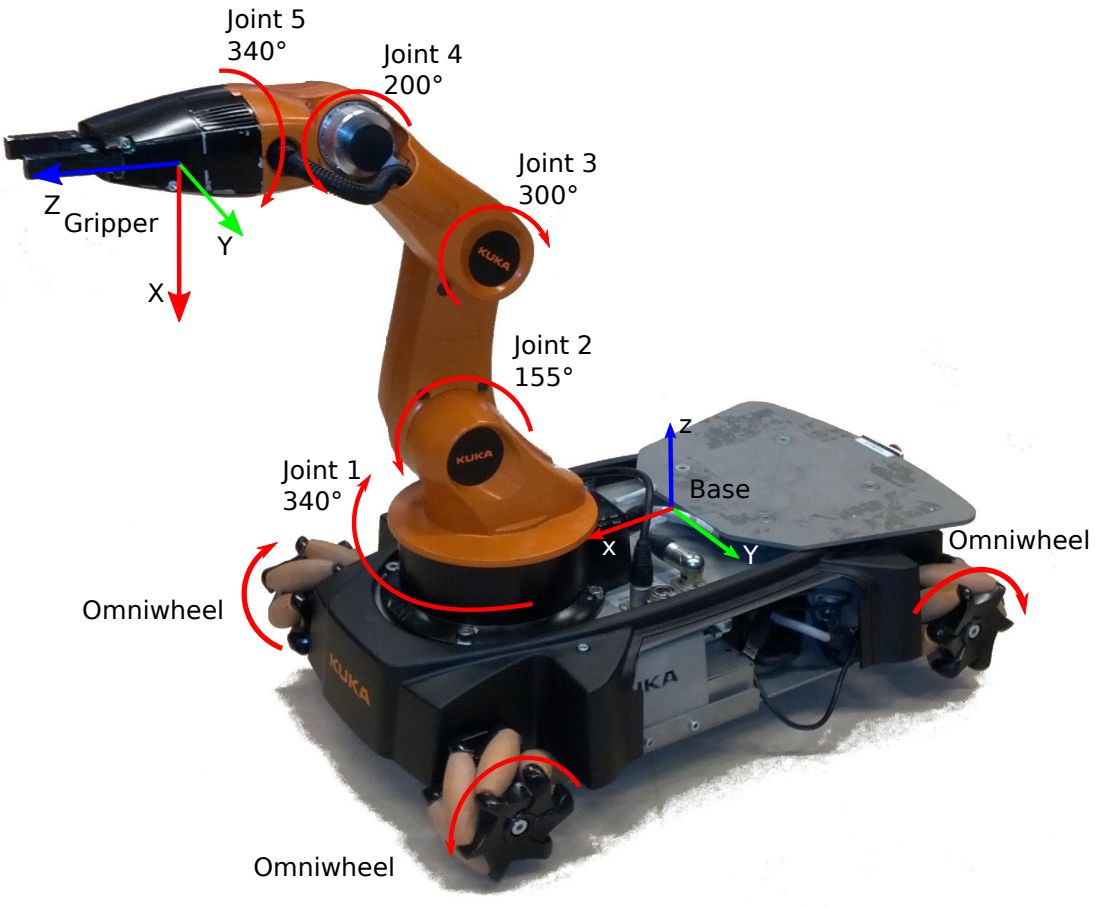

Figure 2.1: The youBot after initialization. The joints 1 to 5 of the robotic arm are given with their range in degrees and their positive direction using an arrow. The wheels are shown with their positive direction of rotation. The reference frame of the gripper and base are shown.

The gripper cannot move freely in all direction in space. For example, in Figure 2.1 the gripper cannot rotate around his x axis without moving in y and z. The robotic arm has 5 degrees of freedom and one depending degree of freedom. Special joints configuration can limit the gripper in more direction, for example when the joints are at the end of their range.

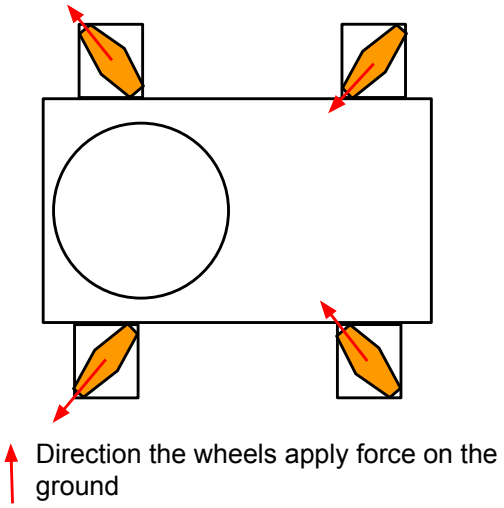

Figure 2.2: Force direction in which the omni-directional platform can apply force to the ground. In the current state the platform would go to the right.

The platform consists of four wheels with their own motor and encoder. An omni wheel has multiple rollers placed under an angle of 45 degrees, due to this configuration they apply a force under an angle of 45 degrees. The combination of the applied force of the wheels, results in the net force on the base. The platform can translate in x and y and rotate around the z axis .

2.1.2 Electrical

The youBot has an EtherCat bus for communication between his the actuators and the embedded computer. Each motor of the arm and base has his own driver for low level control and reading out the motor encoders. The drivers can do PID control on torque, velocity and position. Torque control uses current measurement as feedback. The position and velocity control uses the relative encoders on the motor. The network overview of the EtherCat is given in Figure 2.3.

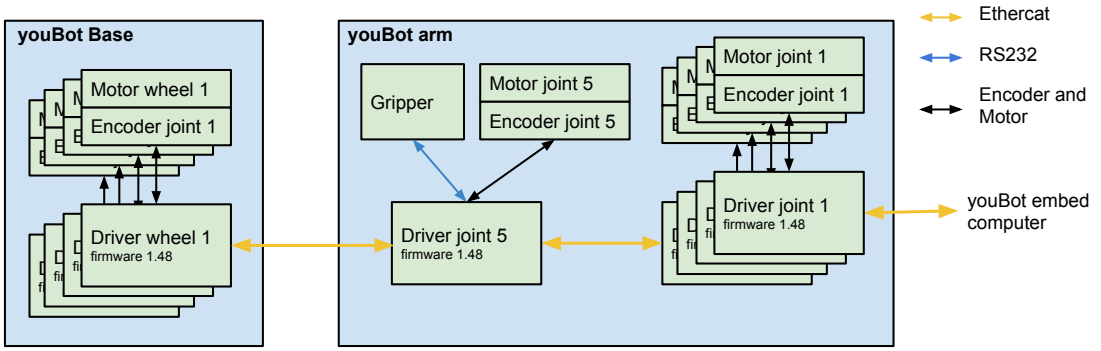

Figure 2.3: Hardware communication of the arm and base. Joint 5 has the gripper attached over a serial connection.

The gripper is not directly connected to the EtherCat bus, it is connected to the motor driver of the 5th joint using a serial connection as shown in Figure 2.3. This driver has to control the motor and forwards the commands from the EtherCat to the gripper by a serial connection.

2.2 youBot software

2.2.1 Control overview

The demo software uses the joystick and haptic device which are connected to the remote computer to control the youBot arm and base. The arm and base of the youBot are separately controlled, using virtual setpoints which are generated on the remote computer. The virtual positions and actual positions of the arm and base are communicated over the network between the youBot and remote computer. The youBot arm and haptic device use an impedance controller to steer the robotic arm and to calculated the haptic feedback as shown in Figure 2.4.

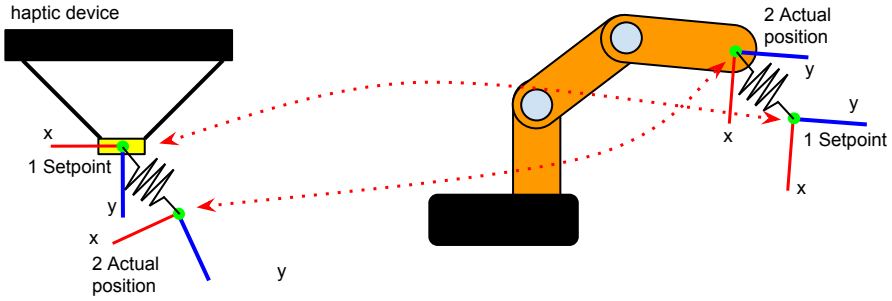

Figure 2.4: Systematic overview of the impedance controller.

2.2.2 Software overview

The remote computer is connected to the joystick and haptic device over USB. The drivers of both device are connected to the setpoint generation software on the remote computer. The setpoint generation software uses the input of the devices to generate the virtual setpoints for the base and arm. The setpoint generation uses the actual position of the youBot arm and the virtual setpoint for the arm to calculate the force feedback of the haptic device.

The youBot receives the virtual setpoints and uses it as the setpoint in the control loops of the arm and base. The output and feedback of the control loops are connected to the drivers which communicate with motors on the youBot. The system overview is shown in Figure 2.5.

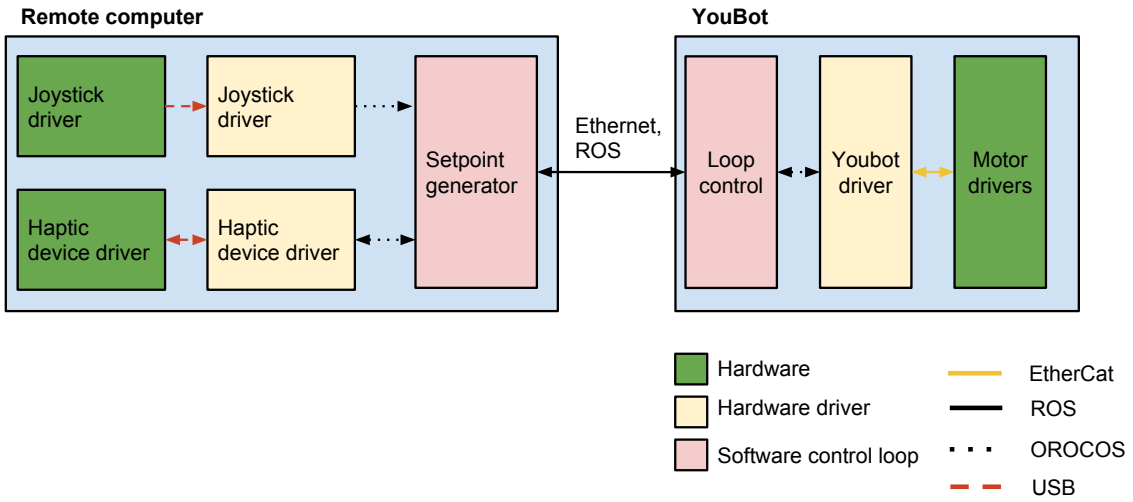

Figure 2.5: youBot demo software overview. The impedance controllers as shown in Figure 2.4 is done in the red block.

2.2.3 Control algorithm

The control algorithm can be split into two main models, the setpoint generation software which runs on remote computer and the youBot control model as shown in Figure 2.5. The youBot control algorithm is modelled for re-usability, it is called the "motion stack" and is one of the outcomes of the BRICS project (Brodskiy et al., 2011). The motion stack controls the youBot arm from the actual position to the virtual setpoint using an impedance controller. An overview of the control structure is presented in Figure 2.6.

The setpoint generation software is redesigned during this project, information on setpoint generation is given in 3.2. The control of the youBot, the motion stack, is mostly reused. Some background information on the motion stack will be given next. The motion stack is split up in multiple block as shown in Figure 2.5. The motions stack is split up in blocks to make it reusable for other robotic arms. The motion stack is designed and tested using 20-sim and directly implemented using the code generation from 20-sim with an OROCOS template. Each block is exported as a separated OROCOS software package. The separation is software package makes it possible to run them in parallel such that the multiple processing cores on the embedded computer in the youBot can be used efficiently. Each block within the motion stack will be further explained below Figure 2.5.

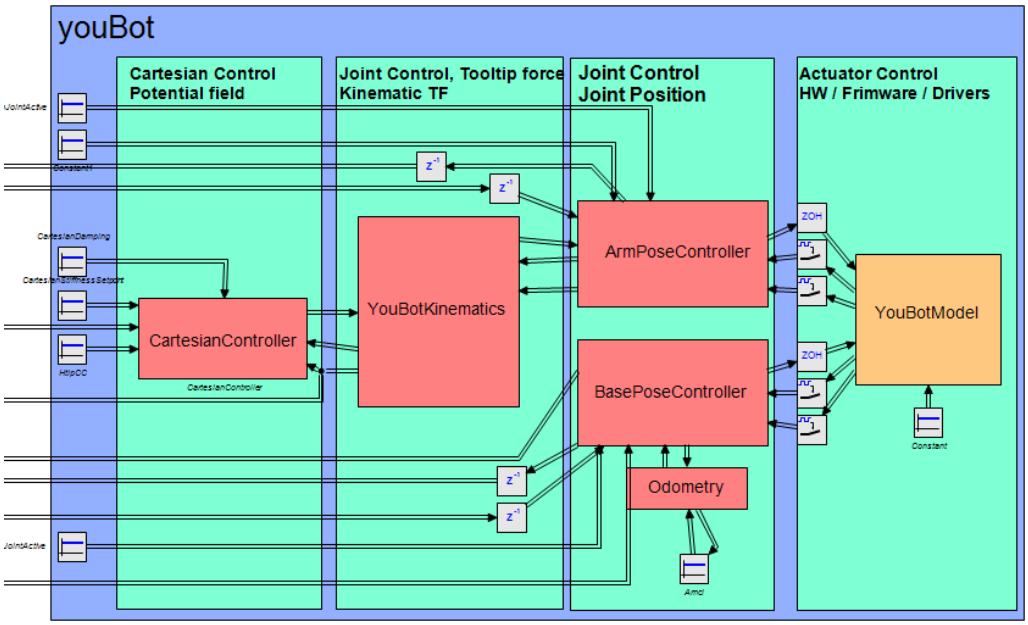

Figure 2.6: 20-sim Model of the youBot. The red block contains the loop control in the youBot in Figure 2.5. The signal form the left come from the remote computer.

The following blocks are shown in Figure 2.6:

Cartesian control "CartesianController"
Creates a virtual spring-damper between a virtual setpoint in space and the tip of the youBot arm. The virtual spring calculates the wrench on the gripper which drives it to the virtual setpoint in space. This block works in Cartesian space.

Kinematics "YouBotKinematics"
Calculates the geometric Jacobian depending on the joint configuration. The geometric Jacobian gives two relations: first between the torque on the gripper and torque on the joints. Secondly it is used for the twist of the gripper and the joint angular velocity.

Base position control "BasePoseController"
The wrench of the base is converted to the torques on the motors. This extra conversion step is needed because of the topology of the omnidirectional platform. The safety layer is included in this model.

Odometry "Odometry"
The angular speed of the wheel is used to track the position. The initial position is used as the reference position.

Arm position control "ArmPoseController"
Arm position control consists of the gravity compensation and safety layers are included in this model.

Driver "Driver"
The driver contains the dynamical model of the youBot. In the generated software it is replaced by the real youBot arm and platform. It has the same interface as the driver software package which is used in the real demo.

For more information on the motion stack used in the youBot demo software please refer to Brodskiy et al. (2013).

Energy passivity layer
The youBot and remote computer contain the passivity layer. The passivity layer is a safety layer, it makes the system stable even when the connection to the haptic computer is bad or lost. It is done by giving energy packages to actuators which they need for moving. New energy is supplied by the remote computer. If the connection is lost, no new energy will go to the youBot. This will stop the actuators when the local energy buffer is depleted. Sudden movements will drain the energy buffer, stopping the youBot safely.

There are three parameters on which the passivity layer is depending: initial energy, size of the energy buffer and the energy supply rate. Setting these parameter too low prevents the arm from moving, too high values makes the energy passivity layer useless. The accuracy of energy estimation determinants how good the energy passivity layers works. Please refer to (Brodskiy et al., 2013) for more information on the passivity layer.

2.2.4 Code generation

The 20-sim model is translated to OROCOS software packages using the 20-sim code generation. The software packages run on the youBot and remote computer. Figure 2.7 gives an overview of the demo software packages with special attention if it is generated using 20-sim or programmed by hand using ROS or OROCOS. The software structure is based on the 20-sim structure as shown in Figure 2.6 and Figure 2.7. The 20-sim generated software package uses a different data structure for communication compared to the drivers packages. Therefore a special conversion layer are designed in OROCOS, they are shown in purple.

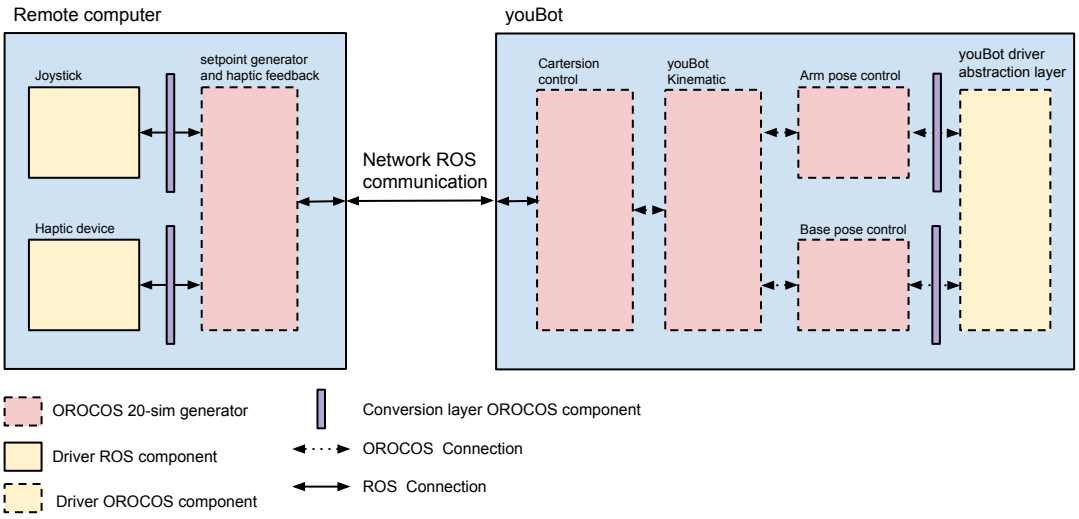

Figure 2.7: youBot software overview. Right is based on the 20-sim model in Figure 2.6.

There are extra packages which provide service to the demo software like setting the control parameters. These packages are explained on the RaM wiki site (Wiki, 2014b), they are not shown in Figure 2.5.

2.2.5 Software driver

The youBot computer has a driver for communicating to the motor drivers over EtherCat, the youBot_driver packages. The youBot computer driver is wrapped in the youBot_OODL packages such that it can communicate with OROCOS packages as shown in Figure 2.8.

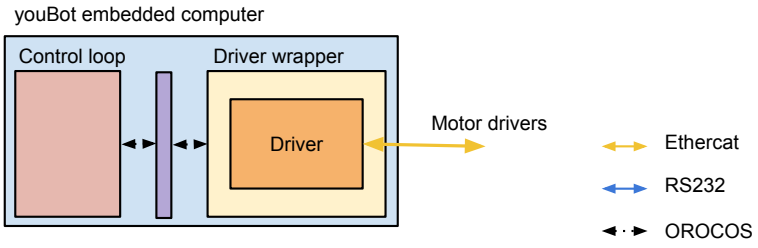

Figure 2.8: youBot software overview.

2.3 Toolkit overview

For the demo software different software library's and design software are used. A short overview on the used tools is given.

2.3.1 20-sim

20-Sim is a simulation package to simulate and design dynamical models and controllers. A complete model of the youBot and controller is made using 20-sim. Changes in the controller can be tested using 20-sim simulation before trying them on the youBot.

2.3.2 OROCOS

OROCOS is a library which can be used for advanced machine and robotic packages. It has a real-time implementation for creating packages which have thread-save and lock-free communication ports. An external scheduler can be used to schedule the packages execution during runtime. OROCOS can only be used for soft real-time behavior, a special operation system should be used to make it hard real-time.

XML files can be used for configuration within OROCOS packages. The use of XML files makes it possible to make parameter changes without recompiling. This is useful when fine-tuning parameters of controllers has to be done. The deployment of OROCOS packages is done using OROCOS operation files.

2.3.3 ROS

ROS is a framework for creating robotic applications. It allows software packages to have harmonized interfaces making them reusable in other applications. The problem with ROS is that it has no real-time performance. The main reason it is used in the demo software is that it allows packages to be connected over Ethernet. Special tools for plotting data and showing reference frames during runtime are available within ROS. This makes model verification and debugging possible.

The BRICS project developed a ROS development kit, called Bride. Bride is Eclipse based and provides a graphical interface for modelling software packages. It has the option to do code generation to ROS and OROCOS packages using Python or C++ code. The Bride integration with the used toolchain is outdated and broken, it is not used during this project.

2.4 Toolkit integration

The software library OROCOS, the framework ROS and simulation software 20-sim are integrated into one toolchain. Documentation on the design of the toolchain integration is given in Brodskiy et al. (2013).

In the original toolchain Bride was integrated for designing and configuring communication. The 20-sim integration with Bride is not up to date with the current version of Bride. Therefore the communication needs to be configured manually without the help of Bride.

2.4.1 OROCOS 20-sim integration

20-Sim has a code generation function which is template based. The toolchain consists of a template for 20-sim to export OROCOS sofwate packages. The template creates a XML file containing all the parameters and port information. The XML file is loaded on initialization of the OROCOS package, this makes parameter tuning possible without recompiling. The OROCOS ports are equal to the interface of the exported 20-sim model. Dynamic equation computations and connections within the 20-sim model are exported to C++ code. Within the C++ generated code the 20-sim library codes for matrix, integrations and angles conversion are used. The "update hook" in OROCOS is used to execute the 20-sim generated code. Each update is equal to a simulation step in 20-sim. A tutorial on how to use the template can be found at Wiki (2014a).

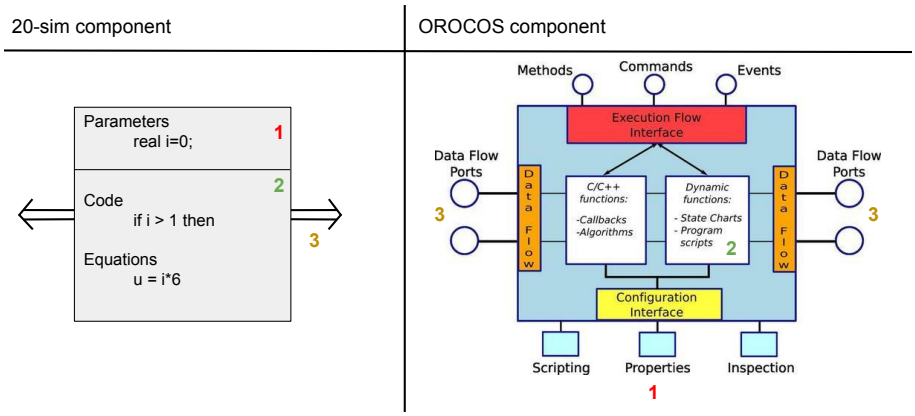

Figure 2.9: From a 20-sim packages to a OROCOS packages using the 20-sim code generation. One and three are loaded using on initialization using the XML file. Two, the dynamic model is converted to C++ code.

2.4.2 ROS and OROCOS

The integration of OROCOS with ROS allows OROCOS packages to communicate with the ROS packages. This allows the OROCOS packages to be used for real-time behavior and ROS to be used for communication and debugging. The connection between ROS and OROCOS packages is configured using an OROCOS operation file which is loaded using the OROCOS ROS integration.

2.4.3 20-sim ROS integration

ROS uses standard data structures for communicating reference frames, twists and wrenches. These are different from the 20-sim data structures in the generated code. 20-Sim generated code needs a conversion layer to communicate standardized ROS communication messages. The conversion layers are designed in OROCOS and connect the 20-sim generated OROCOS packages to convert the 20-sim data structure to ROS data structure. The use of the standardized ROS communication messages makes it possible to use ROS visualizations tools for debugging.

3 Demo upgrades

The goal of the project is to upgrade the user interface. The background information on the youBot will be used in this chapter. The first step in implementing the new user interface was adding an extra safety layer. This was needed to protect the arm from damaging itself during testing. The design of the extra safety layer is described in 3.1. In 3.2 and 3.3 the renewed base and arm user interface respectively are presented.

3.1 Safety features

The old youBot demo software has safety features included in the control algorithm. The most important one is the passivity layer, other safety features are the mechanical joint endstop protection of the arm and motor driver signal clipping which protect the motors from overcurrent.

During this project, it became clear that these safety features where not sufficient to protect the youBot arm against damaging itself. The youBot arm damaged itself during testing. In this chapter, it will be explained what the limitation of the safety features are and which features are added to protect the youBot arm from damaging itself.

3.1.1 Problem description

The passivity layer is depending on the initial energy and supply rate and energy buffer size. These parameters became incorrect during the project. The initial energy was to high and the energy supply rate to low. This was caused by changes to the input of the remote computer and decoupling the base from the arm controller. The incorrect parameters made the passivity layer useless. Two extra safety layer are added to protect the joints of the arm.

The second safety feature is the mechanical end stop protection. This safety feature protects the control algorithm from hitting the mechanical end-stop of the joint. The control algorithm uses a non-linear virtual spring which starts to counteracts the driving force when the joint is near its endstop. This should prevent the motor from hitting the endstop. The counter force, created by the virtual spring, was summed to the control signal as shown in Figure 3.1. The problem with the old setup was that torque joint driving signal was not limited, changes to the Cartesian impedance controller could complete overrule the virtual spring. The endstop protection should be updated in such way that it is not influenced by changes made to the input.

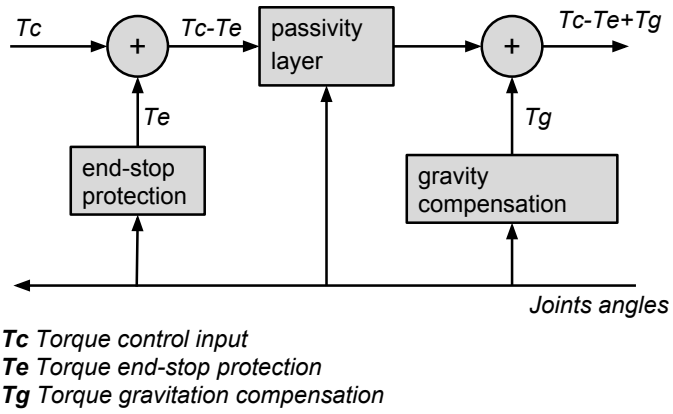

Figure 3.1: How the safety feature where implemented in the arm pose control (Figure 2.7).

3.1.2 Design

In this section, the design of the new safety features are described. The new safety feature are a torque limitation and a velocity limitation. These safety features are designed to be more robust against changes in the control algorithm.

Torque limitation

Torque limitation is added to protect the motor gearbox and limits the accelerations. Torque limitation can be applied on the wrench off the gripper in Cartesian space or directly on the joint of the youBot arm. Limiting the the gripper torque in Cartesian space does not guarantee torque limitations in the joints. The main goal is to protect the gearbox of the joints, therefore torque limitation in the joints is implemented. The joint control torques are cutoff when they override the threshold value. The torque limitation is done before the endstop protection, this makes the maximum driving torque at the endstop predictable. This ensures that the mechanical endstop protection cannot overrule the driving force when the stiffness of the controller is changed. An extra torque limitation is done after endstop protection, passivity layer and gravitation compensation, to ensure these layers does not make the driving torque too high as shown in Figure 3.2..

Velocity limitation

Velocity limitation prevents the robotic arm to move at high velocity. The Kinetic energy is quadratic depended on the velocity of the arm. Hitting something at a high velocity creates a high energy impact which is dangerous for the robotic arm and environment. The velocity limitation is implemented as joint angular velocity limitation. Torque control with angular velocity limitation can be applied in multiple ways. Lowering the torque if the joint reaches the maximum angular velocity will not ensure that the joint actually slows down, joints do influence each other and can speed up other joints. This is especially a problem with wild movements of the arm. Therefore non-linear damping is used, the damper starts counteracts the driving force when a joint reaches its maximum threshold velocity. The torque limitation before the endstop and velocity limitation ensures that the velocity limitation force can overrule the control force. Both safety layer are shown in Figure 3.2

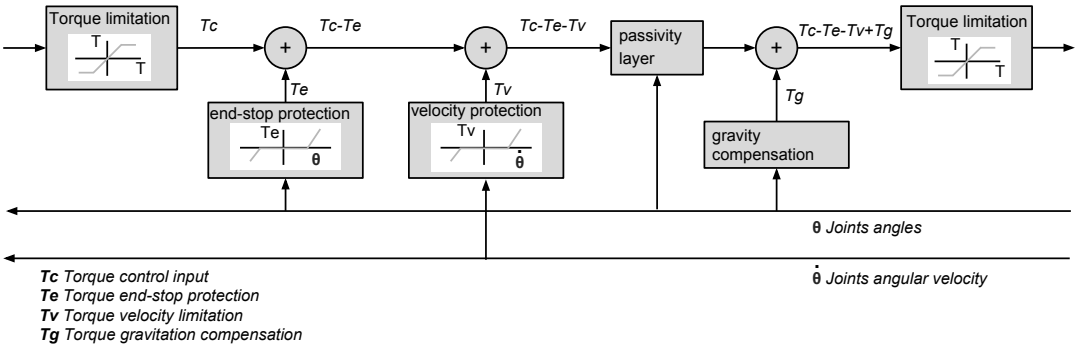

Figure 3.2: New safety design within the arm pose control. The non linear damping and non linear endstop protection are given in the small plots within the software block

During the project there was a problem with implementing the velocity limitation. The angle of the joints where numeric differentiated to get the angular velocity. This did not work due problems with the scheduling order of software package. To solve this, the angular velocity of the motor drives is used. More information about this can be found in Appendix A.

Endstop protection

The torque control limitation helps the endstop protection, by making the maximum control torque predictable. This ensures that the virtual spring can always cancel out the driving force. Stopping the driving force does not slow down the arm, the arm can still hit the endstop at the set maximum velocity limitation. To prevent this the vitual spring is setup in such way that it stops the arm before hitting the endstop. The virtual spring has to counteracting the driving force and store the kinetic energy of the arm.

A parameter estimation is done to get an idea on how the virtual spring should be setup to protect the arm from hitting the endstop. The parameter estimation is done for the worst case situation. The first joint is used because it drives the most mass, the rest of the arm is set complete stretched as shown in Figure 3.3. In this configuration the arm can store the most kinetic energy. The torque and angular velocity limitation are set respectively to 10Nm and 1 rad/s to make the worst case situation predictable.

First the virtual spring has to completely cancel out the driving force, this stops the arm accelerating in the direction of the endstop. This should take place within an angle of 0.1 radial. The torque control limitation is set to 10Nm, to complete cancel out the control torque, the virtual spring should deliver 10Nm in 0.1 radians. This makes the virtual spring constant 100Nm/rad. Secondly the kinetic energy of the arm needs to be transferred to the virtual spring. The kinetic energy of the arm is giving in Equation 3.1.

$$E_k = \frac{1}{6} M l^2 \dot{\theta}^2 \tag{3.1}$$

The virtual spring "stores" energy as given in Equation 3.2.

$$E_{spring} = \frac{1}{2} k \theta^2 \tag{3.2}$$

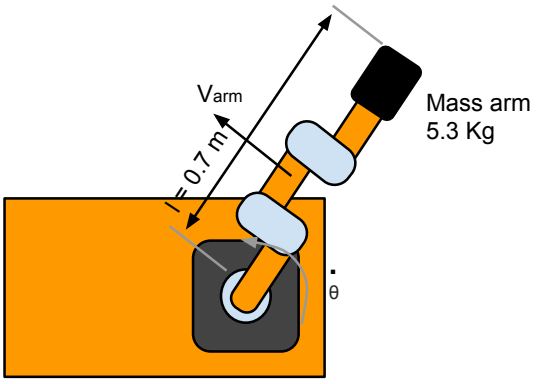

Figure 3.3: Calculation of the endstop using the first joint. Assumed that the arm is fully stretched.

The used spring constant of 100Nm/rad is used to calculate the angle were the energy of the virtual spring is equal to kinetic energy of the moving arm. Using the parameters from the youBot arm given in Figure 3.3, results in an angle of around 0.1 radians.

Next is checked if the endstop protection does not exceed the maximum driving torque. At an angle of 0.1 radians the virtual spring stays below the maximum control torque limitation of 10Nm. The virtual spring should start counteracting the driving force 0.2 radians before the endstop with a spring constant of 100Nm/rad.

In this estimation some dynamical behaviors are not taken into account. For example the arm could retract itself speeding up the angular velocity around the first joint. It is unlikely that the joint would reach the mechanical endstop at a high velocity. Hitting the endstop with some velocity is not problematic as long the motor already pushes away form its mechanical endstop when it hits the mechanical endstop. In this situation there will be no high torque change in the gearbox and motor axis. This is guaranteed by the limitation in the control torque and the stiffness of the virtual spring.

3.1.3 Implementation

The implementation is done within 20-sim. The use of 20-sim makes it possible to test the control algorithm offline using a simulated model of the youBot. The OROCOS package is created using the 20-sim code generation when the control algorithm works correctly in simulation. The generated package can simply replace the old OROCOS package.

The modular design of the motion stack makes it possible to do specific changes without influencing other parts of the demo software. All the safety features are build into the "ArmPoseController" package. Each safety layer is tested in 20-sim and added to OROCOS.

3.1.4 Testing

Debugging and visualization tools are used for debugging and testing the software code. ROS has multiple tools which allow to visualize information, for example the values of software packages ports. The tools can only be used on ROS ports, this means that the 20-sim OROCOS generated code needs to be connected to a ROS port. For debugging ROS facilitates multiple visualization tools like plotting port values. It should be taken into account that ROS package ports keep their last value when there is a latency in the network. This is shown in plots as a horizontal lines. The latencies can make it hard to check realtime behavior or find noise sources.

The angular velocity limitation is tested using the haptic device to steer the youBot arm. The results are plotted in Figure 3.4. The velocity limitation threshold is set to 0.7 rad/s with a damping constant of 30Nm. The velocity limitation needs an overshoot to cancel out the control force. At a angular velocity of 1 rad/s the velocity limitation completely cancels out the control torque.

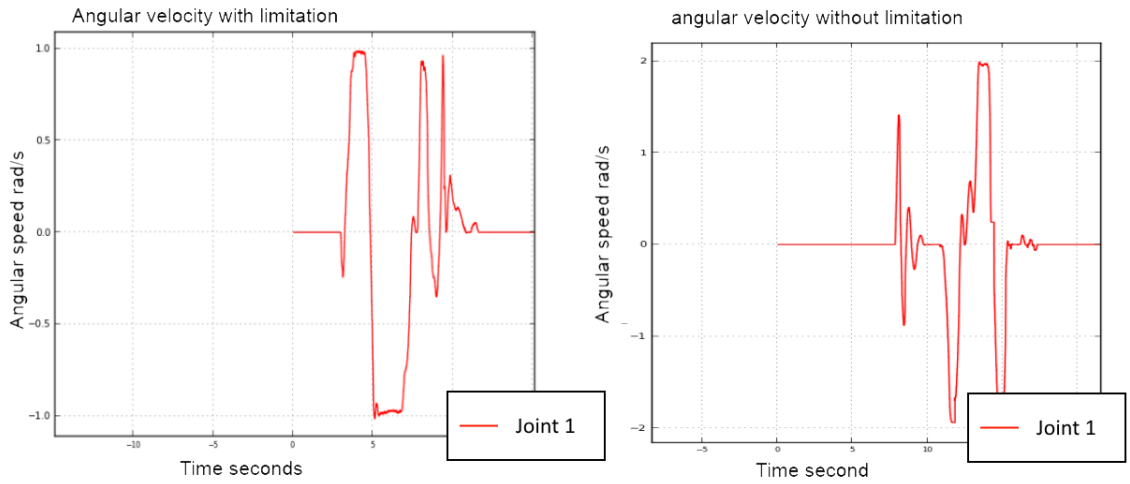

Figure 3.4: First joint angular velocity plotted with and without velocity limitation. The haptic device is used to steer the robotic arm.

3.2 Joystick integration

The old demo used the haptic device connected to the remote computer for controlling the youBot base and arm. The base moved in the direction of the arm if the arm could not reach the setpoint. This gave limited control to the base, for example the base and arm could not move in different directions. Splitting the arm and base control, using the joystick for the base and haptic device for the arm, will give more control to the base. A three axis joystick is connected to the remote computer which is used for controlling the stiffness of the youBot arm impedance controller and for energy injection in the passivity layer.

3.2.1 Design

The first design step is defining how the input of the joystick is translated to movement of the youBot base. The simplest design is to do a one-on-one mapping of directions, in the way that the youBot moves in the direction in which the joystick is pushed, as shown in Figure 3.5. The control frame is fixed to the base, this means that the user has to rotate the control in the direction of the youBot base. This could be solved by mounting a camera on the youBot and streaming it to the remote computer, such that the user can use the screen as a reference.

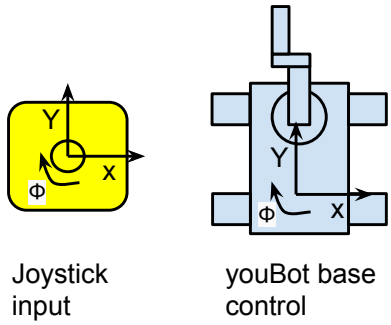

Figure 3.5: From joystick movement to base movement.

There are multiple control algorithms which can be used to control the youBot base in the direction of the joystick. Direct force and position control are tested to see how they work in a real world situation.

Force control

Direct force control uses the joystick input to apply a wrench to the base. The youBot base controller can translate a wrench on the base to the equivalent torques on the wheels. The base accelerates in the direction of the applied wrench. The base stops accelerating when the friction is the same as the applied wrench, or the motors reach their maximum speed. Direct force control means that the youBot has to be manually stopped by applying a counter force. When the connection is lost, the last applied wrench stays on the youBot. This smooths out a short connection lost, but makes it dangerous when the connection is lost for a longer period. The passivity layer will be used to stop the applied wrench when the connection is lost too long.

Force control is a kind of feed-forward control. Feed-forward control cannot take into account small frictions differences in the wheels or other unmodeled dynamical behavior. The other problem is that the center of mass of the youBot is not precisely known. This center of mass is needed for calculating the torque on the wheel to match the applied driving wrench. This can result in the youBot starts rotating around his axis when it should only moves sideways. The user can counteract the rotation around the base using the joystick.

Position control

Position control is more complex compared to simple feed-forward force control. Position control uses the joystick to create a virtual position on the remote computer which is send to the youBot as a setpoint. The youBot has an odometer to track his actual position. The odometer position and virtual setpoint position is used in a control loop to calculate the wrench which is needed to go from the actual position to the virtual position.

The virtual setpoint is calculated using the joystick input as twist expressed in the base frame.

$$T_b^{b,0} \tag{3.3}$$

The twist is multiplied with the current position of base frame to get the positional changes. This is integrated in time to get the virtual position.

$$T_b^{b,0} = H_0^b \dot{H}_b^0 \tag{3.4}$$

$$H_b^0 T_b^{b,0} = \dot{H}_b^0 \tag{3.5}$$

$$H_b^0 = \int \dot{H}_b^0 dt + H_b^0(0) \tag{3.6}$$

An advantage of position control is that the control loop of the base will counteract when the youBot base is pushed away from the virtual setpoint position. This can for example happen when the robot arm pushes against a wall.

On a connection latency the youBot will stop at his last received virtual setpoint position. During the timeout the setpoint on the remote computer keeps moving in the direction where the joystick is pushed in, there is no feedback to the remote computer setpoint generation. The result is that on connection restore the virtual setpoint and actual setpoint have a large gap. The gap between the setpoints gives in a large control action in the base position control loop. The result is that the base sprints to the virtual setpoint after the connection is restored, this makes large timeouts dangerous during operation. The passivity layer is not ideal to solve this problem. The correct energy estimation of the remote computer is from the previous virtual setpoint to the next virtual setpoint. This energy estimation is not enough to close the gap between the setpoint and actual position when the connection is restored. This stops the base safely but leaves a gap between the virtual setpoint and actual position, which makes the base difficult to control. The energy estimation would be correct if the actual position of the youBot is used on the remote computer to calculate the estimated energy. A correct energy estimation will not stop the base sprinting to the virtual setpoint position.

The passivity layer cannot be used for dealing with the network latency. The setpoint generation itself should take into account that there are latency in the network. Therefore the actual position is send as feedback and is used for limiting the virtual setpoint. The virtual setpoint position stops moving when it is to far away from the actual position. The complete position control loop is given in Figure 3.6.

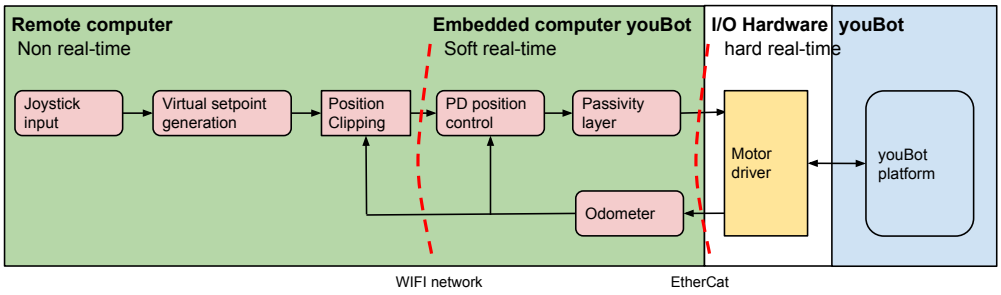

Figure 3.6: The base control loop with timing guarantees. The unreliable WIFI connection is between the remote computer and the youBot.

The passivity layer prevents the control algorithm, which steers the base from the actual position to the virtual setpoint position, from becoming unstable. This can happen when the youBot drives over a floor which gives less damping, or when the wheels are lifted from the ground. The joystick has only one output, this cannot be used to express power supply rate directly. The calculated twist from the joystick input is used to calculate the energy supply rate which is linear depending on the input. This is transferred to the youBot base using the energy transportation protocol.

3.2.2 Implementing

In the first iteration, the youBot base and arm are decoupled, this means that both can be controlled separately. Within the motion stack, the base and arm are seen as a kinematic link, the base is modeled as the first joint which has three degrees of freedom. The geometric Jacobian is changed such that it represents the kinematic chain from gripper to the youBot base instead of from the gripper to the ground. This has some basic consequences in the structure of motion stack, for example the calculated torques do not need to be split up into the base torque and arm torque.

The joystick implementation uses a standard ROS package for communicating with the hardware layer. A conversion layer in OROCOS is used to convert the data-type from the joystick structure to 20-sim twist structure. The joystick is implemented using a ROS package. The ROS package implementation means that the input is not real-time. This is not problematic because the joystick has no force feedback and most of the latency of the joystick input to the control loop is created by the wireless network.

Force control implementation

Force control is implemented by directly connecting the joystick conversion layer to the base controller which is running on the youBot. The conversion layer already had a feature build in to amplify the joystick input, this is used to tune the applied wrench such that it was high enough to move the base. The old passivity layer used the haptic input for energy supply. This is no possible anymore because the arm control with the haptic device is decoupled from the base control. There is a simple energy supplier available, which can be used to set a constant supply rate. The simple energy supplier is fine-tuned such that it gives enough energy to the base during normal operation. The youBot base energy tank threshold is set to a level that it will drain fast enough on a connection lost and does not influence normal control.

Position control implementation

Position control without feedback was already implemented in the 20-sim model. The position controller was disabled after the initialization by a service package, this was updated such that it did not disabled the position control. The virtual setpoint form the remote computer is reconnected to the "baseposecontroller" on the youBot and the actual position is send as feedback back to the remote computer. Both connection are done over the network using ROS.

The setpoint generation is limited such that it cannot move too far away from the actual position. The virtual setpoint has to stay within a virtual circle of the actual position. The virtual setpoint stops moving when it is outside the virtual circle. This only works for translation directions. The rotation is limited to a maximum angle difference. Both limiters work in Cartesian space, this gives their limiting values real world limits, like how many meter the youBot base can move when the connection is lost.

3.2.3 Testing

Position and force control are both tested. The energy passivity layer was switched off during the initial testing phase. Small blocks, which lifted the youBot wheels from the ground, were

used such that the initial testing could be done safely. During testing it became clear that the wireless network was more unreliable than expected. Network latency were measured using the ping command from the remote computer to the youBot. Latency of one second were not unusual during operation.

Force control testing

The force control works with some limitations. The first limitation is that the youBot tends to rotate around his axis when it is moving sideways. The model does not show this behavior, there is an unmodeld dynamical behavior on how the wheels rotations applies force on the ground. The control does not corrects for this because the controller is feed-forward. The user can try to counteract the rotation when the base is moving sideways. Both scenarios are tested using the joystick as input, the absolute angle is plotted in Figure 3.7. The user can counteract the rotation a bit, but higher network latency can make this very difficult.

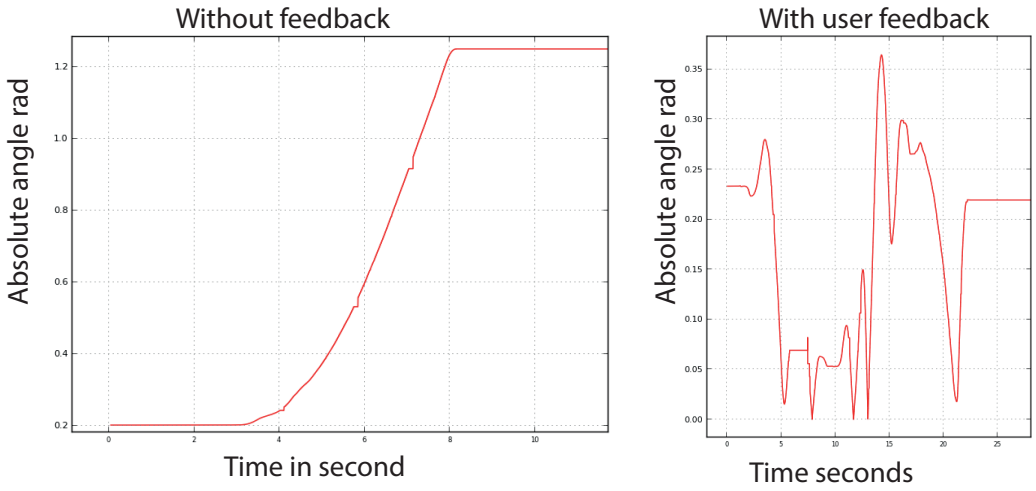

Figure 3.7: Moving the base side way using the joystick and feed-forward force control. Absolute angle around the Z axis referenced from the youBot base initial pose.

The second limitation is that the platform is difficult to stop due to the high latency in the wireless network. The passivity layer, which is used to make the platform safe against network latency did not work. The passivity layer only stops the driving force, the youBot base keeps moving in the direction in which ts was moving. The passivity layer only makes the base passive in the direction in which it is moving. Force control could work if the wheels have more friction such that the base stops faster when the passivity layer stops the control force. The low friction in combination with the unreliable network make this control dangerous to use.

Position control testing

The position controller on the youBot needed some fine tuning to get it working correctly. Moving sideways needed a larger control gain compared to moving forward. This has probably to do with the unmoddeld dynamical behavior in how the wheels rotations applies force on the ground.

On the remote computer the joystick input is scaled to set the maximum speed of the virtual setpoint. The circle which limits the setpoint was made big enough such that it does not disturb normal control in the presence of small network latencies.

Both problems from the force control are corrected using the position control. The position controller moves the base sideways using the feedback to cancel out the rotation. The feedback shows up as a small oscillation in rotation angle around the z axis as shown in Figure 3.8. The

base is easily stopped by moving the joystick to the center. A high network latency stops the youBot until the connection is restored.

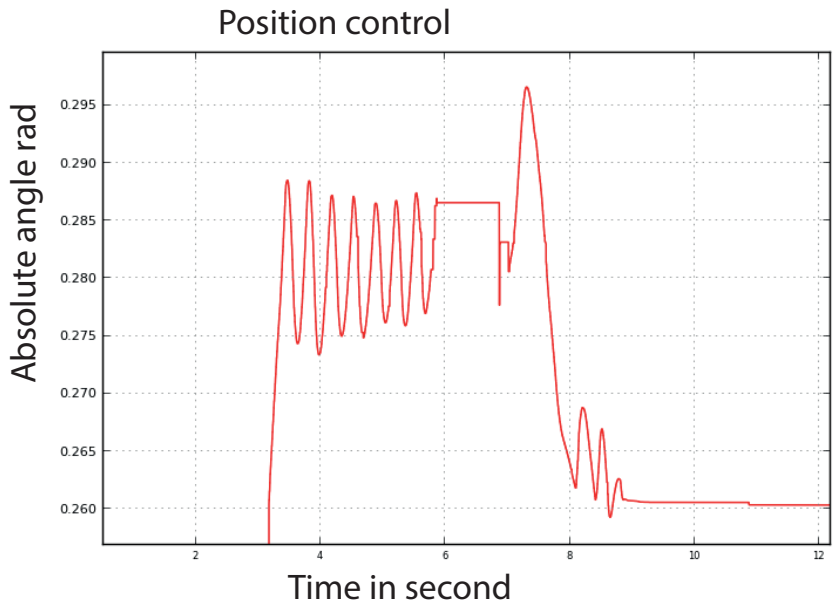

Figure 3.8: Moving the base side way using the joystick and position control. Absolute angle around the Z axis referenced from the youBot base initial pose.

3.3 Haptic input

There is a new haptic device available, the Omega 6. The Omega 6 is better designed and can create a stronger and more accurate feedback force. The haptic device driver is integrated into a software package such that it can communicate with the 20-sim generated software package and the haptic device.

The remote computer and youBot communicate the virtual setpoint and actual position over a network. The youBot has his own control algorithm for controlling the gripper from his actual position to the virtual setpoint send from the remote computer. The remote computer uses the actual position of the gripper to calculate the haptic feedback. Due to this setup the setpoint generation can be updated on the remote computer without changing the youBot control algorithm.

The haptic input is used on the remote computer to generate a setpoint for the gripper. The setpoint generation on the remote computer is updated, keeping the limitation of the new haptic device into account. How this is done will be explained in this chapter.

3.3.1 Design

The haptic input design is split up in four sub designs.

- Setpoint generation
- Scrolling
- Haptic feedback
- Initialization

Each design will be treated separately, explaining which problems had to be dealt with and how it is solved.

Setpoint generation

The old setpoint generation does not take into account that the robotic arm cannot move freely due to topological limitation of the arm. The gripper rotations are limited by the topology of the robotic arm. The main limitation of the gripper is that it almost always cannot rotate around his X axis, for example in the initialization pose. The user needs to set a setpoint using the hapic device which is reachable for the gripper. The haptic feedback should help by giving feedback to the user when the setpoint is not reachable. The problem is that the haptic device has no rotational feedback. The user has to rotate the setpoint such that it is reachable by the gripper without the guidance of the haptic feedback. This is difficult to do.

This is solved by coupling the rotation to the translation in setpoint generation. The rotation of the setpoint is made partly a dependent state. The setpoint is rotated in such way that the Z axis of the setpoint goes through the center of the youBot arm base, as shown in Figure 3.9. This creates a setpoint around the base of the arm which is rotational reachable for the gripper.

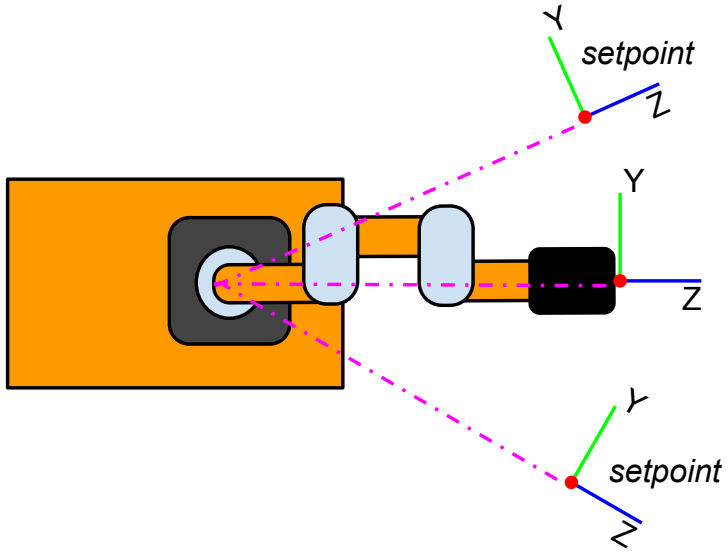

Figure 3.9: A systematic top view of the youBot base with arm. The setpoint is rotate depending of the position such that the Y axis is pointing to the base of the arm

The gripper can freely rotate around his Z axis and most of time around his Y axis. These rotations are added after the reachable setpoint is generated. The last two axis of the haptic device are used for controlling these orientations as shown in Figure 3.10. This means that one of the axis of the haptic device is not used.

Figure 3.10: Overview how the haptic device controls the youBot arm. The rotation around the Y and Z of the gripper are directly controlled by the tip of the haptic device as shown with dubble arrows in the Figure. The transnational direction are controlled in Cartesian space referenced from the youBot base.

The gripper and haptic device have different aligned reference frames. This can be solved by coordination transformation such that the haptic is aligned with the youBot gripper frame after initialization. The haptic feedback has to be transformed back to the haptic coordination frame.

Scrolling

The youBot arm has a larger workspace compared to the haptic device. The haptic device can only control the robotic arm in small part of the workspace of the robotic arm. This is partly solved by scaling the output of the haptic device. The new haptic device has an accuracy of

<0.01mm making it accurate enough to scale the position output. The setpoint translation is scaled three times, scaling the position output more makes the control of the robotic arm difficult. Therefore scrolling is added such that the limited workspace of the haptic device can be moved over the workspace of the youBot arm.

The setpoint scrolling is, like the setpoint generation from the haptic input, split into translation and rotation. The translation setpoint scrolling starts when the haptic device reaches his limitations. The translation limitations is detected by using an ellipse to mark a virtual border. The setpoint is translated using a rate which is depending on how much the haptic device passes the virtual border of the workspace of the haptic device. The virtual border has to be smaller then the actual workspace limitation to give some space for the scrolling input. For more information please refer to Ruesch et al. (2012).

The haptic device has a large enough rotational input such that scrolling is not needed. Implementing can easily be done in joint space, by translating the angle when the haptic input is near its rotational limitation.

Haptic feedback

The haptic feedback is important for controlling the robotic arm. The haptic device can only give feedback in translational direction, this is taken into account by the modification of setpoint generation as described in previous section 3.3.1.

The haptic feedback was calculated using a six dimensional spring. Because of this, it is possible that some of the feedback is calculated as a rotation, which can not be displayed by the haptic device. Therefore I switched to a three dimensional spring which only calculates a translational feedback.

The Haptic feedback uses the passivity layer. Energy is supplied to the youBot arm when it is pressed against the haptic feedback.

Initialization

The youBot arm goes to his initialization pose on start-up. During the initialization of the remote computer the haptic device is set to the zero position using the haptic feedback. This makes the virtual setpoint zero, which is in the base of the arm. The setpoint generation software should start with sending the actual position of the youBot arm, which is defined from the base of the arm. To achieve this the actual initial position is added to the virtual setpoint. This makes the first generated virtual setpoint of the position of the gripper. This prevents wild movement of the arm when the remote computer connects.

Passivity layer

The Haptic feedback is essential for the passivity layer to work correctly. The passivity layer uses energy pairs for calculating the energy use or supply to the passivity layer. For example, a joint on the arm uses 20 joules if it is rotated 2 radians using 10 newton meter. The remote computer adds energy to the passivity layer if the user presses against the haptic feedback. For example pressing 10 cm against 10 newton supplies 1 joule to the system.

The frictions in the joints of the youBot arm uses energy while moving. The energy supply rate should be high enough to overcome these joint frictions. A too low energy supply rate lowers the control torque, which results in a larger gap between the virtual setpoint and actual position of the youBot arm. This creates a larger feedback force to the user, which also result in a larger energy supply rate to the passivity layer. The user feel frictions of the youBot arm as drag in the haptic device.

The passivity layers has two energy buffers: on the remote computer and the youBot arm. Energy is supplied and used from these local buffers. The energy buffer are connected to each

other over the wifi using ROS. Energy is exchanged using a simple energy protocol, for more information about this see Brodskiy et al. (2013).

The haptic device only has feedback in the translation direction, this means that for the rotational direction energy needs to be supplied in different way. This is done by calculating the twist created by the rotational change of the input each timestep. The twist is used to inject energy into the energy buffer on the remote computer.

3.3.2 Implementation

During the joystick integration the youBot was already decoupled from the base. Almost all the implementation is done on the remote computer. Only some small changes to the energy distribution and passivity layer are needed on the youBot.

The fist step was implementing the new haptic device as a separated software package. The haptic device has his own library which give a Cartesian control of the haptic device. The driver does all the low level control on the haptic device using a separated thread. The driver is wrapped such that it can communicate with the OROCOS setpoint generation packages. As said, the haptic feedback steers the user to the actual position of the youBot arm. The haptic feedback is calculated in the setpoint generation software package. The feedback will start vibrating if the actual position of the haptic device arrives too late in the setpoint generation or the feedback force is too late. The timing between the haptic device and setpoint generation is critical. Therefore initially is tried to wrap the driver library into an OROCOS package. Due to problems between the library and OROCOS I switched to a ROS implementation. The setpoint and feedback communication is fast enough preventing the user to feel vibrations in the feedback.

The haptic device software package has to communicate with the 20-sim generated software code. The haptic device uses a standardized ROS message structure, a transformer message for the position and a wrench message for the feedback. This has the advantage that ROS tooling can be used to visualize the haptic input. For both messages a conversion layer between the 20-sim and generated code was available and is used.

The setpoint generation is implemented as shown in Figure 3.11. The input is first statically rotated such that it is aligned with the youBot, then split into translational and rotation of the setpoint. The scrolling for translational directions is added to the translational setpoint. The translational setpoint is rotated such it is reachable for the youBot arm. The rotation setpoint is added after the setpoint is made rotational reachable. This makes the orientation referenced from the translational setpoint. This prevents that the direct controlled orientation of the gripper of the youBot changes when arm translates.

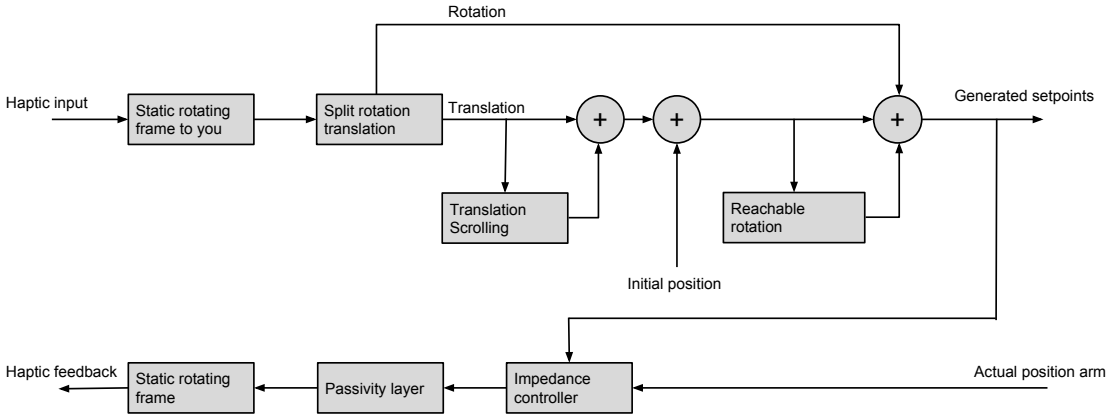

Figure 3.11: Overview how the setpoint generation software is implemented.

3.3.3 testing

During the testing it became clear that some damping is needed to keep the Haptic device stable. Normally the passivity layer should keep the haptic device stable, but during the testing phase the passivity layer was setup incorrectly. Damping is added such that the haptic device is more stable. Here the timing limitation of the ROS implementation of the haptic device became clear. The motors of the haptic device started to generate a sound. The damping could not be calculated from the position of the haptic within the 20-sim model. The noise was due to communication latency. Therefore damping is added to the feedback force within the haptic device driver.

The passivity layer works limited. Without the rotation energy injection the passivity layer worked as expected. The arm could be translated using the haptic device. Rotating the tip used the energy buffer and stopped the arm form further movement. Adding the injection worked but was not accurate enough to set the passivity layer supply rate correctly. The supply is set so high that it takes a while for the energy buffer to be consumed and safely stopping the system. Although the passivity layer works the energy estimation could be better. Currently the system is mostly kept stable by the damping in the haptic device and arm which keeps the impedance control loops stable on the youBot and haptic device.

4 Conclusion

The goal of this project is to split up the base and arm such that they can be controlled separately. A short overview is given how they are updated during the project.

The base of the youBot is controlled using a 3 axis joystick. The control loop is designed to take network latency into account. The separation of the base and robotic arm gives more control to base and makes it possible to demonstrate the functionally of the omnidirectional wheels. The joystick is successfully implemented and can be used to steer the base over the wireless network.

The new Haptic device is integrated into the software. The setpoint generation is updates such that it keeps the rotational limitation of the arm and hatpic feedback limitation into account. The new haptic device is successfully implemented and can be used to control the youBot arm.

An extra safety layers is added for protecting the youBot arm for damaging itself. There is tried to setup the passivity layer correctly, but the lack of rotational feedback made it more difficult then expected. It became clear that the setpoint generation design was more focused in generating reachable setpoint instead of using the passivity layer to keep the system stable. Although the passivity layer is enabled, the damping on the haptic device and youBot arm ensures the stability of both controllers.

During the project, the control algorithm and development toolchain are tested. The control algorithm on the youBot, the motion stack is updated during this project. The motion stack structure helped with making structural changes without losing structure. The software development toolchain which uses 20-sim for modeling and code generation for implementation is tested. The software development toolchain significantly decreased the software development time by automating the implementation phase from 20-sim to the real control software. The tooling from ROS helped in testing en debugging the software updates.

4.1 Recommendation

The following topics should be looked into further

- Getting the passivity layer working correctly.
- Modeling realtime behavior using the toolchain.
- Maintaining the toolchain.

The topics are further explained below.

The current setpoint generation is optimized for generating reachable setpoint. It does not take into account that it needs to supply energy to the system only using the force feedback in the translation directions. This needs to be updated such that a correct amount of energy is supplied to the arm and the setpoint is automatically rotated such that the arm can reached it.

The toolchain makes the youBot an ideal platform for rapid prototyping and testing of complex control algorithms. The current toolchain is limited in how it can model realtime behaviour. The software scheduler can only be used to configure sequential execution, which was not enough for correct execution of the demo software. In the demo software the dynamical ordering of the scheduler resulted into noise of the angelar velocity. The current software solved it by using multiple control loops which have their own timing demands and choosing the correct states to communicate between the packages. The scheduling behavior should be taken into account when working with the toolchain. An other solution would be switching to the Utwente developed toolchain Luna for modelling the realtime control algorithm. Luna

should be integrated into ROS such that ROS can be used for non realtime demanding tasks, like debugging and path planning.

ROS is a rapidly evolving framework. This means that the toolchain which integrates 20-sim with ROS should also be maintained. The current toolchain is still usable but maintenance is needed to keep it working in the future. The current toolchain contains the OROCOS template for 20-sim which generates OROCOS software packages. The generated OROCOS package are build using the old ROS building system. In newer versions of ROS has a new building system which is more flexible. The template should be updated to support the new ROS building system. The integration of the toolchain with the ROS development kit Bride is outdated and broken. This should be fixed by updating the plugins which integrates the generated OROCOS components into Bride.

A Appendix: Velocity limitation debugging

During the testing phase it became clear that there was a problem with the joint velocity limitation. The joint driving torque was oscillating and had a lot of noise. The oscillation in the joint driving torque was clearly visible in the joint angular velocity as an oscillation. The first guess was that it had to do with measurement noise in the angles, which is amplified by numeric differentiation to the angular velocity. The angular velocity is used to calculate the damping force which limits the angular velocity. A low-pass filter was used to filter the joint angular velocity. This was an improvement but the result was not good enough as shown in Figure 4.1.

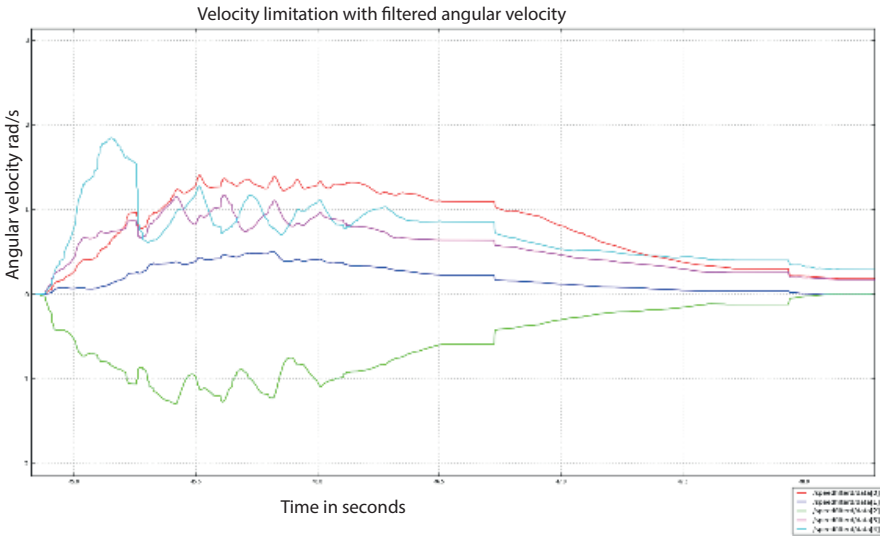

Figure 4.1: The angular velocity of the joints during initialization using the differentiated angular velocity for limiting the speed. The angular velocity is oscillating around the maximum set angular velocity.

The ROS plotting tools are used to get an idea where the noise is coming from. The angular plot showed no noise. Therefore, the numeric differentiation to angular velocity was checked. The numeric differentiation is depending on time. The step-time is set as a fixed value instead of measured during runtime. This can be problematic if the measurement time variates in a timestep.

However, the noise seemed to have a higher variance then could be expected from the variance in the software scheduling. Therefore an extra block is added in the 20-sim model which calculates the current angle minus the previous angle. The resulting plot is shown in Figure 4.2. It is clearly visible that the values are switching between the angular velocity and zero. This is caused by the scheduling order of the software packages. The OROCOS software packages: Driver and drive to 20-sim Conversion packages are needed to be executed to get new angle data available in the arm pose control package. The sequence of the execution is not specified and variates. It is possible that the ArmPoseControl is scheduled first in a timestep. If this happens no new angle data is available in the ArmPoseControl. This makes the numeric differentiation, angle minus the previous angle, zero.

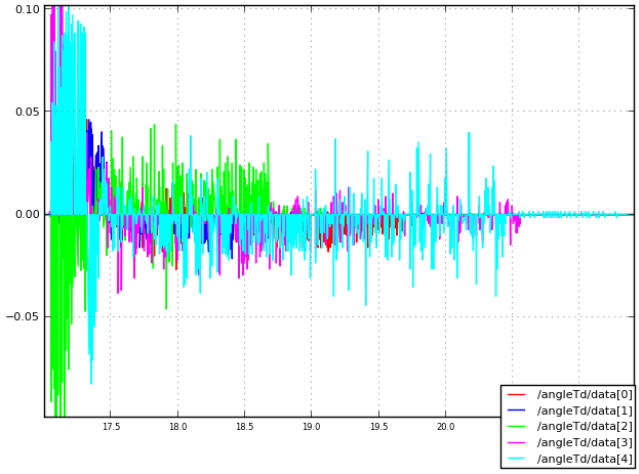

Figure 4.2: Angles differentiated during the initialization of the robotic arm. The values switch between zero and overshoots. This due the scheduling problems between the packages.

To get a better results I switched to the angular velocity which is given by the motor drivers. This solves the incorrect angular velocity due to unscheduled software packages by communicating the angular velocity. Misscheduled software packages now use the angular velocity of the previous time step, instead of using the value zero. An other advantage is that the drivers are hard real-time, making the angular velocity measurement more accurate.

The communication of the angular velocity from the motor driver to the 20-sim generated code was already implemented. The 20-sim model needed to be expanded with a extra port to receive the angular velocity data. internally the model needed to be changed such that it used the angular velocity from the motor drivers.

It is also possible to configure the packages such that they are scheduled sequential. This gives a predictable control. However sequential control cost more CPU time due the fact that the 4 core CPU cannot be used to its full potential.

Bibliography

Bischoff, R., T. Guhl, E. Prassler, W. Nowak, G. Kraetzschmar, H. Bruyninckx, P. Soetens, M. Haegele, A. Pott, P. Breedveld, J. Broenink, D. Brugali and N. Tomatis (2010), BRICS - Best practice in robotics, in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pp. 1–8.

Brodskiy, Y., D. Dresscher, S. Stramigioli, J. F. Broenink and C. Yalcin (2011), Design principles, implementation guidelines, evaluation criteria, and use case implementation for robust autonomy, Technical Report D6.1, The BRICS Project (Grant Agreement Number: 231940), available from http://www.best-of-robotics.org/.

Brodskiy, Y., R. J. W. Wilterdink, S. Stramigioli and J. F. Broenink (2013), Collection of methods for achieving robust autonomy, Deliverable FP7 BRICS Project (231940) D6.2, University of Twente.

Ruesch, A., A. Mersha, S. Stramigioli and R. Carloni (2012), Kinetic scrolling-based position mapping for haptic teleoperation of unmanned aerial vehicles, in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 3116–3121, ISSN 1050-4729, doi:10.1109/ICRA.2012.6224744.

Wiki, R. (2014a), RaM Wiki OROCOS ROS 20-sim Toolchain.
https://wiki.ce.utwente.nl/index.php/OrocosToolchain

Wiki, R. (2014b), RaM Wiki youBot demo software overview.
https://wiki.ce.utwente.nl/index.php/YouBotSoftware