KUKA youBot Hardware Interfaces

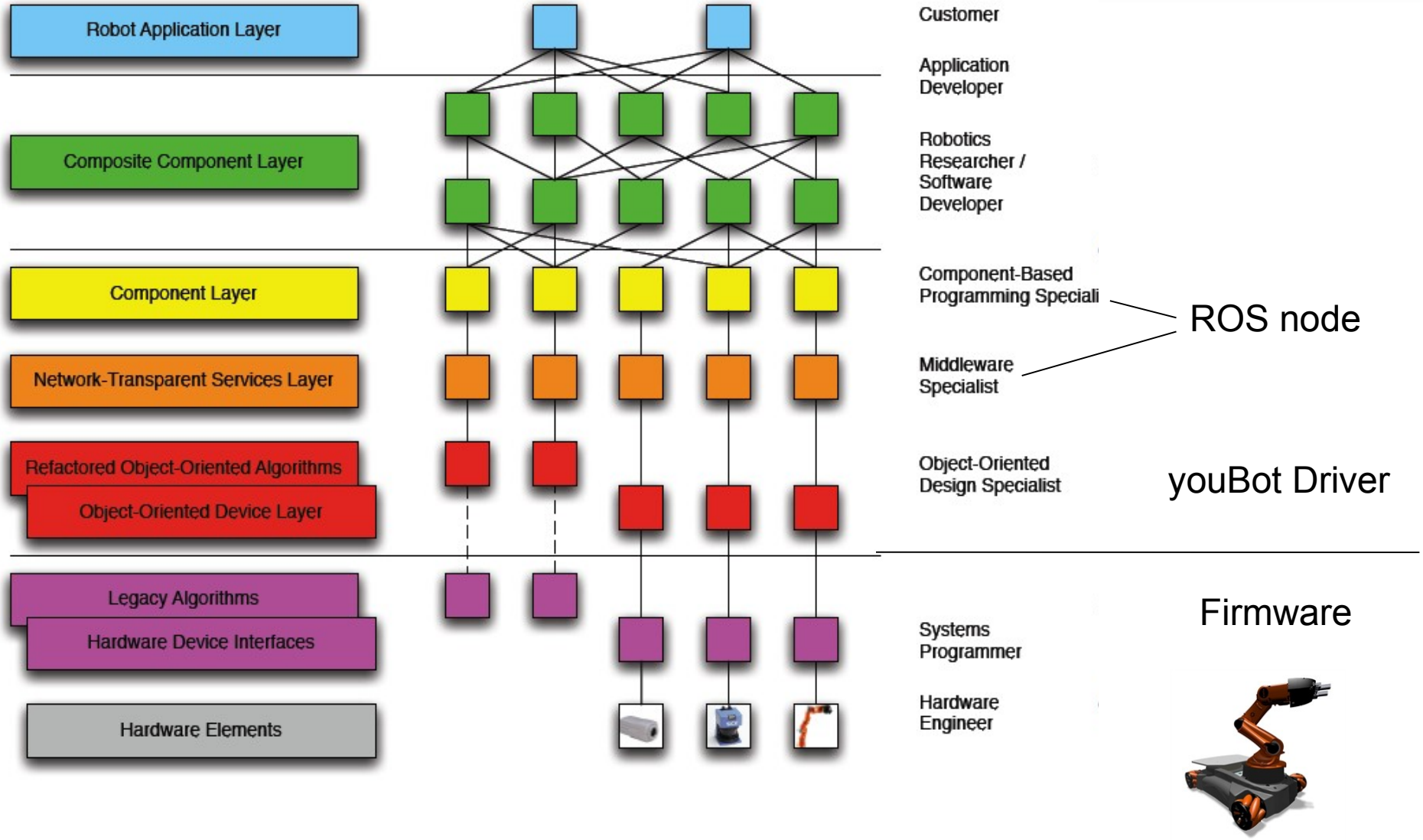
Jan Paulus
Bonn-Rhein-Sieg University

Hochschule
Bonn-Rhein-Sieg

youBot hardware

- omni-directional mobile platform
- 5-degree-of-freedom manipulator
- 2-finger gripper
- all joints with relative encoders
- real-time EtherCAT communication
- on-board PC with embedded CPU,
  2 GB RAM, 32 GB SSD Flash, USB

Abstraction Layers



YouBot Motor Controllers

Each joint has its own motor controller, wich contains:

- ARM Cortex-M3 microcontroller
- Hall Sensors
- EtherCAT Interface
- Position, Velocity, Current PID-Controllers
- I2t monitoring

Firmware Controllers

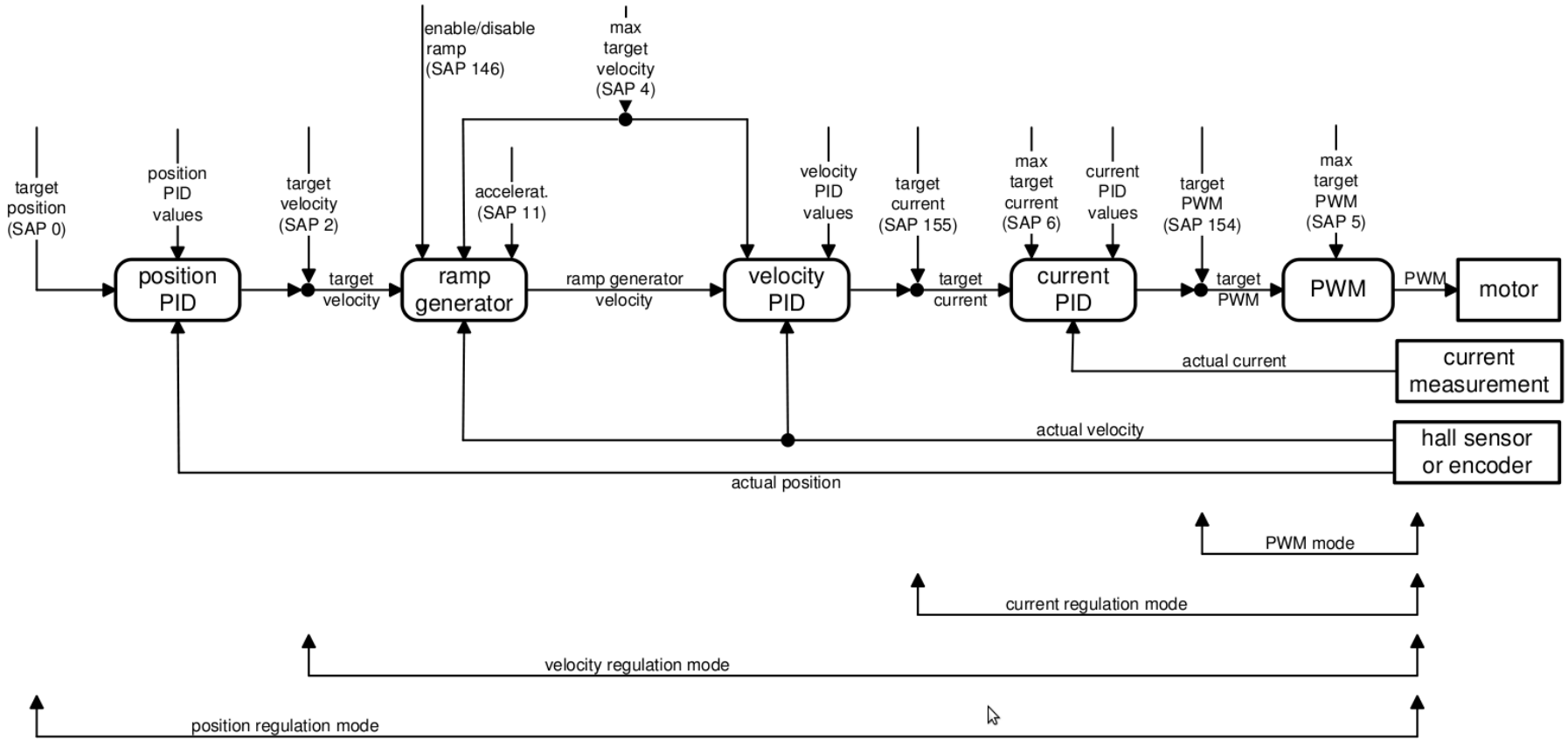
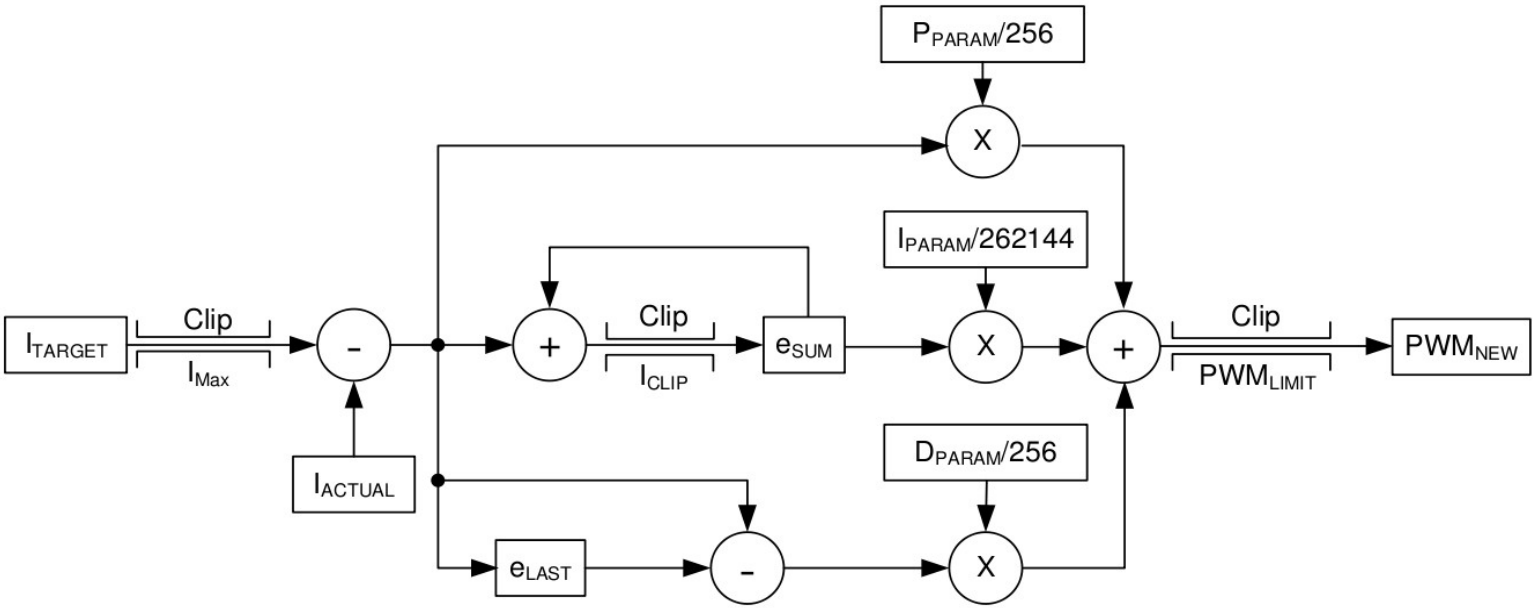


Figure taken from TRINAMIC TMCM-1632/TMCM-KR-841 EtherCAT Manual (V2.3 / 2011-Dec-12)

Current Controller



I_{ACTUAL} Actual motor current

I_{TARGET} Target motor current

I_{Max} Max. target motor current

e_{LAST} Error value of the last PID calculation

e_{SUM} Error sum for integral calculation

P_{PARAM} Current P parameter

I_{PARAM} Current I parameter

D_{PARAM} Current D parameter

I_{CLIP} Current I-Clipping parameter

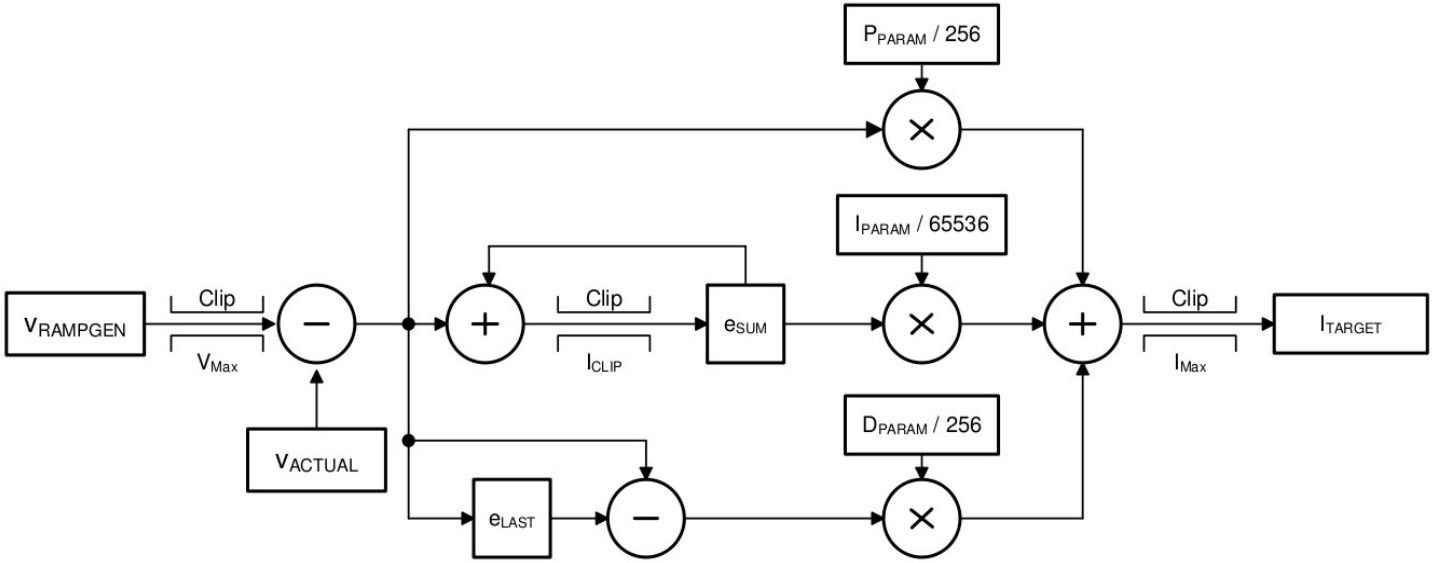
[1/10%] of max PWMLIMIT (a value of 1000 allows the I-part to reach the PWMLIMIT)

PWM_{LIMIT} PWM Limit

PWM_{NEW} New target PWM value

Figure taken from TRINAMIC TMCM-1632/TMCM-KR-841 EtherCAT Manual (V2.3 / 2011-Dec-12)

Velocity Controller



v_{ACTUAL} Actual motor velocity

$v_{RAMPGEN}$ Target velocity of ramp generator

v_{Max} Max. target velocity

e_{LAST} Error value of the last PID calculation

e_{SUM} Error sum for integral calculation

P_{PARAM} Velocity P parameter

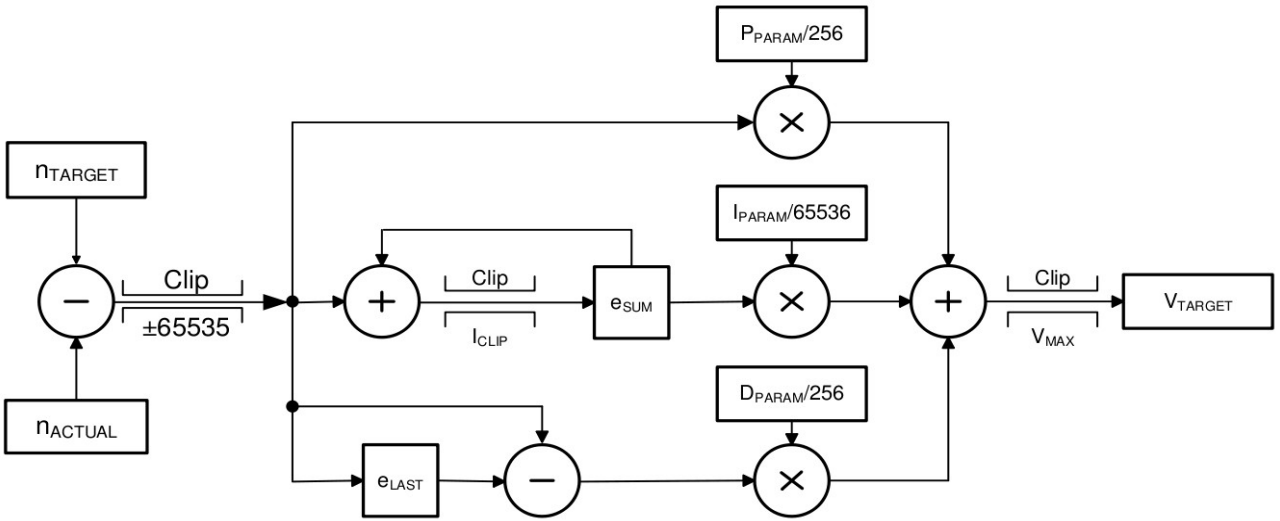
I_{PARAM} Velocity I parameter

D_{PARAM} Velocity D parameter

I_{CLIP} Velocity I-Clipping parameter

Figure taken from TRINAMIC TMCM-1632/TMCM-KR-841 EtherCAT Manual (V2.3 / 2011-Dec-12)

Position Controller



n_{ACTUAL} Actual motor position

n_{TARGET} Target motor position

e_{LAST} Error value of the last PID calculation

e_{SUM} Error sum for integral calculation

P_{PARAM} Position P parameter

I_{PARAM} Position I parameter

D_{PARAM} Position D parameter

I_{CLIP} Position I-Clipping parameter [1/10%] of V_{MAX} (a value of 1000 allows the I-part to reach V_{MAX})

V_{MAX} Max. allowed velocity

V_{TARGET} New target velocity for ramp generator

Figure taken from TRINAMIC TMCM-1632/TMCM-KR-841 EtherCAT Manual (V2.3 / 2011-Dec-12)

Positioning algorithm

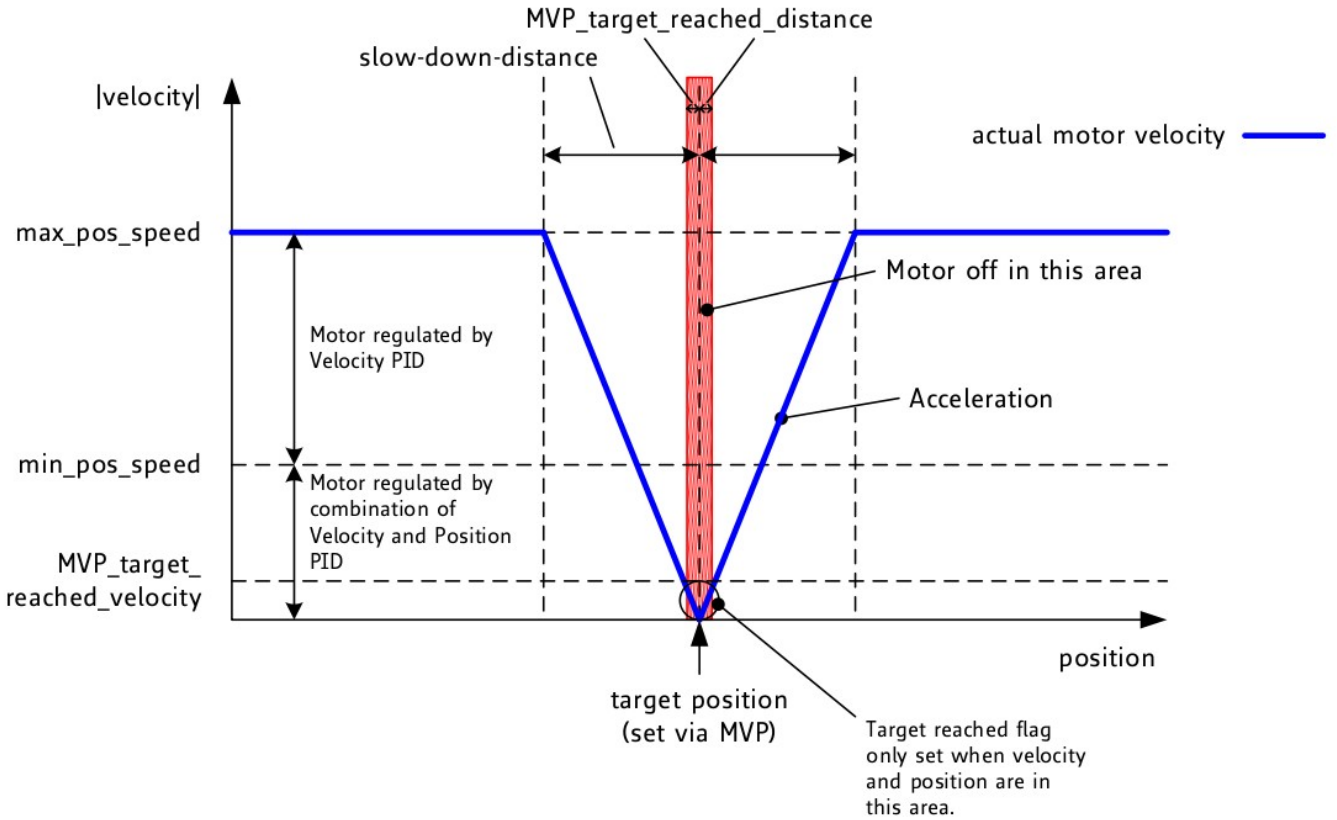


Figure taken from TRINAMIC TMCM-1632/TMCM-KR-841 EtherCAT Manual (V2.3 / 2011-Dec-12)

PID-Parameters

Every PID regulation provides two parameter sets, which are used as follows:

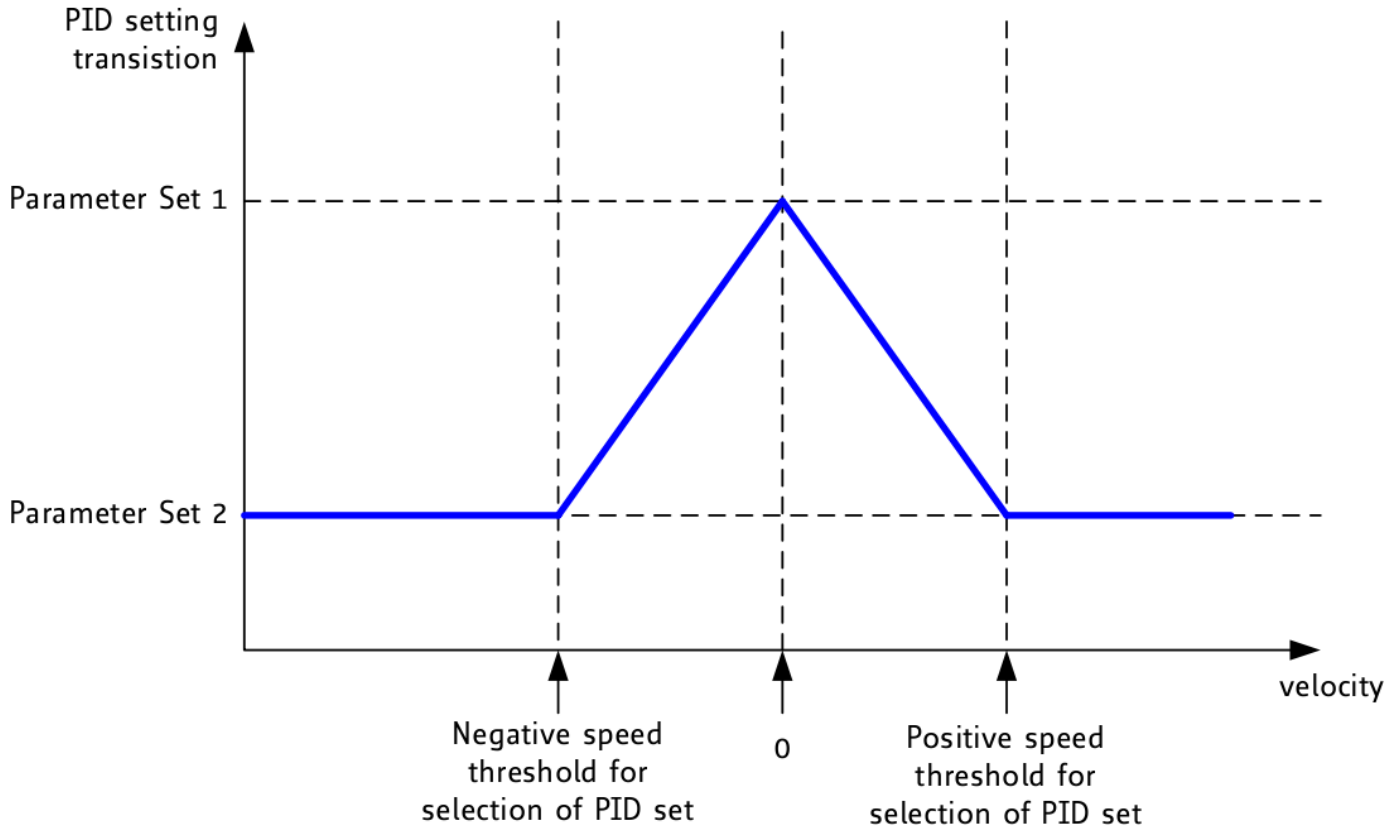


Figure taken from TRINAMIC TMCM-1632/TMCM-KR-841 EtherCAT Manual (V2.3 / 2011-Dec-12)

Safety Features

I2t monitor:

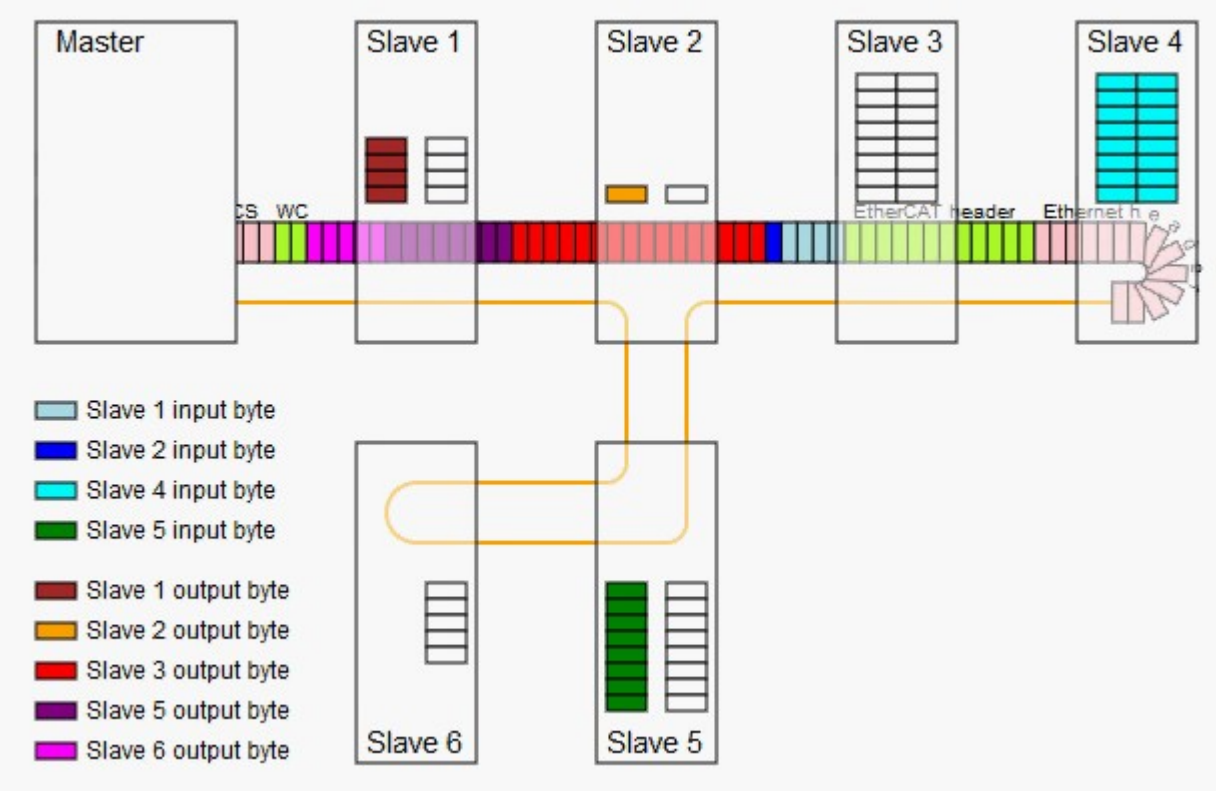
- The I2t monitor determines the sum of the square of the motor current over a given time.
- If the I2t limit is reached the motor will stop and a I2t exceed flag will be set. The user need to unset the flag to continue.

EtherCAT timeout:

- If the EtherCAT communication exceeds a timeout all motors will stop. The user need to unset the EtherCAT timeout flag to continue.

There is NO protected to prevent the manipulator to hit the joint limits.

EtherCAT



- Ethernet fieldbus
- Short cycle times can be achieved

youBot API

- Open source C++ API for joint level control
- Provides full access to the firmware functionality
- API encapsulates EtherCAT communication
- Supported OS: Linux
- Framework independent (ROS and Orocos wrapper available)

What can you do with the API?

Command and sense joint values:

- Position
- Velocity
- Current
- PWM

Configure all joint parameter
e.g. PID controller parameters

Move the base in cartesian space

What can you NOT do with the API?

- Command 6 DOF cartesian position for the manipulator
- The API does not include a forward or inverse kinematic
- Command joint trajectories (wip)

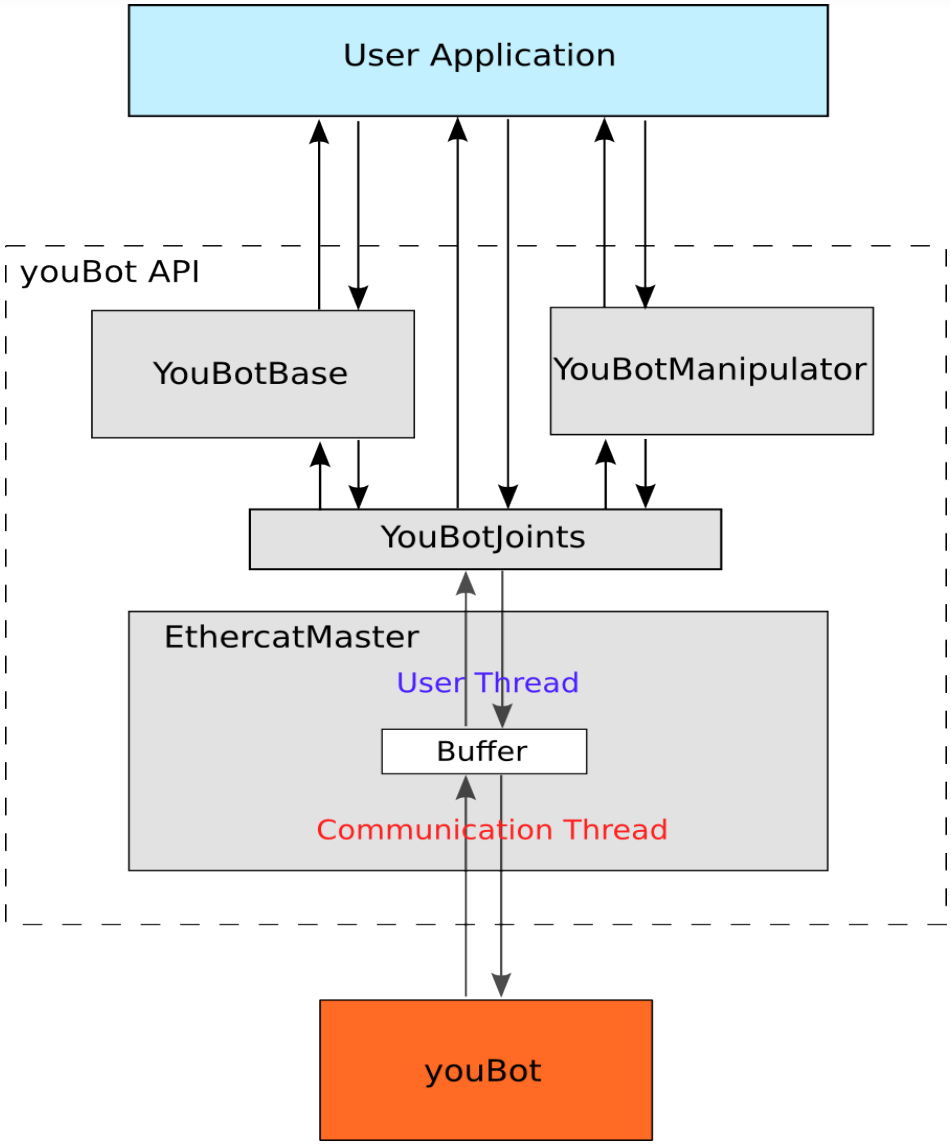
Properties of the youBot API?

- Object-oriented
- Reusable – not lock in to a framework like ROS or OROCOS
- Easy to use
- Communication details are hidden from the user
- Try to minimize implicit assumptions:
  - physical units are represented
  - value ranges are made explicit
  - there are no assumptions about the timing or the usage
  - platform assumptions are avoided

Properties of the youBot API?

- interfaces are used to do abstraction
- stateful interfaces are minimized
- separation of constance
  - setData() used for data flow
  - setConfigurationParameter() used for configuration
- Each configuration parameter is one class
  - by inheritance it is easy to set multiple parameters at once

youBot Driver overview



Timing

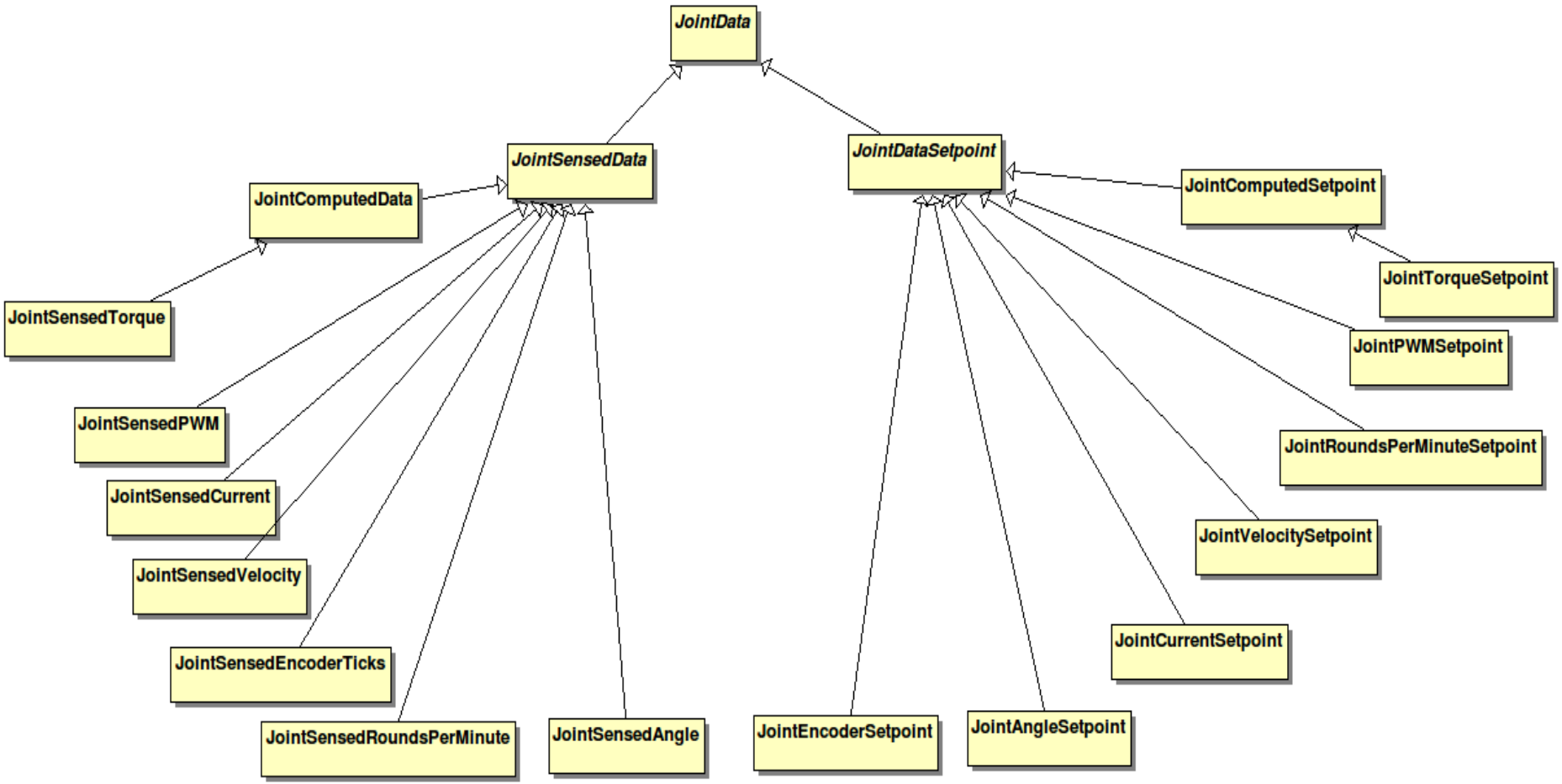
API with communication thread

- default cycle time is 1 ms
- jitter is about 25 microseconds

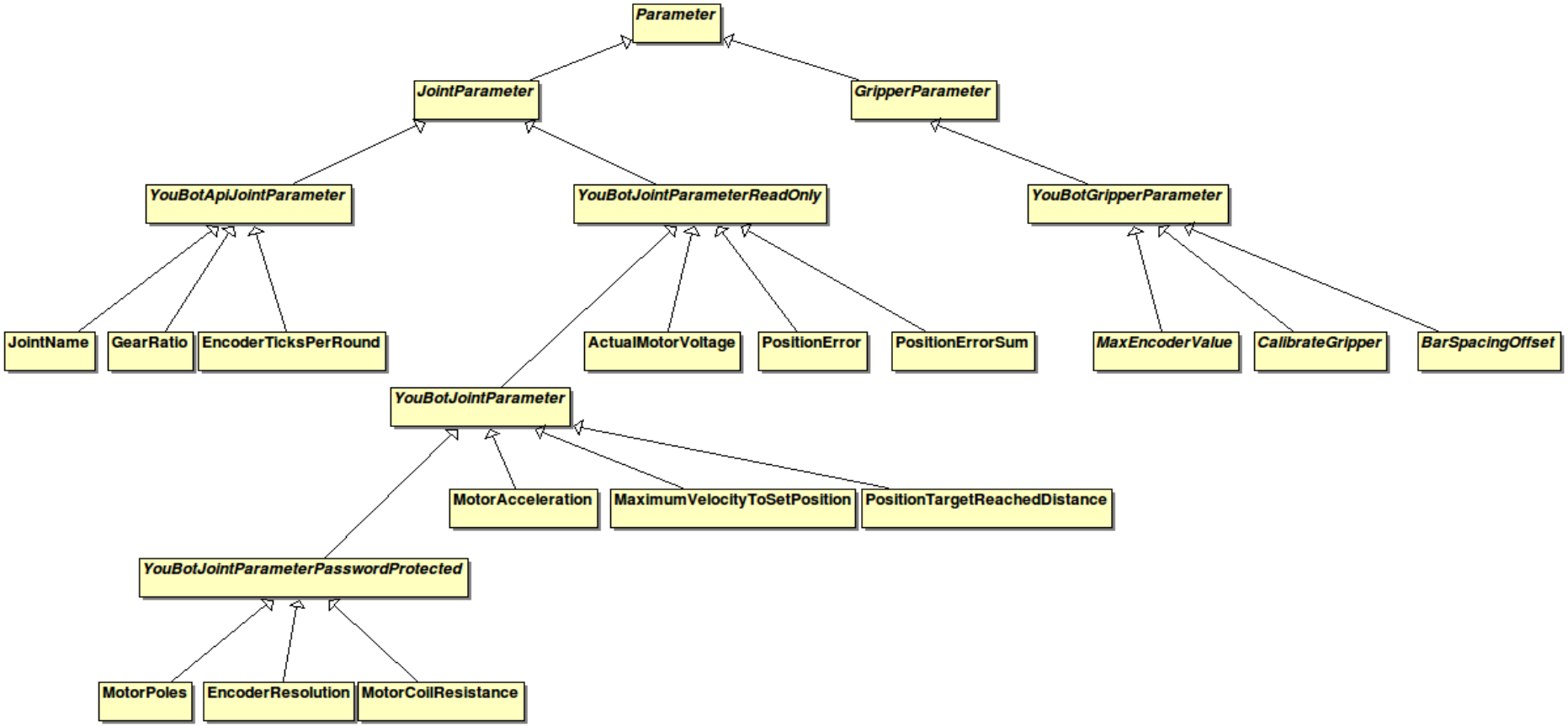
Without a communication thread

- the user has to call sendProcessData() and receiveProcessData() to trigger the communication
- This could be scheduled by a real time operating system

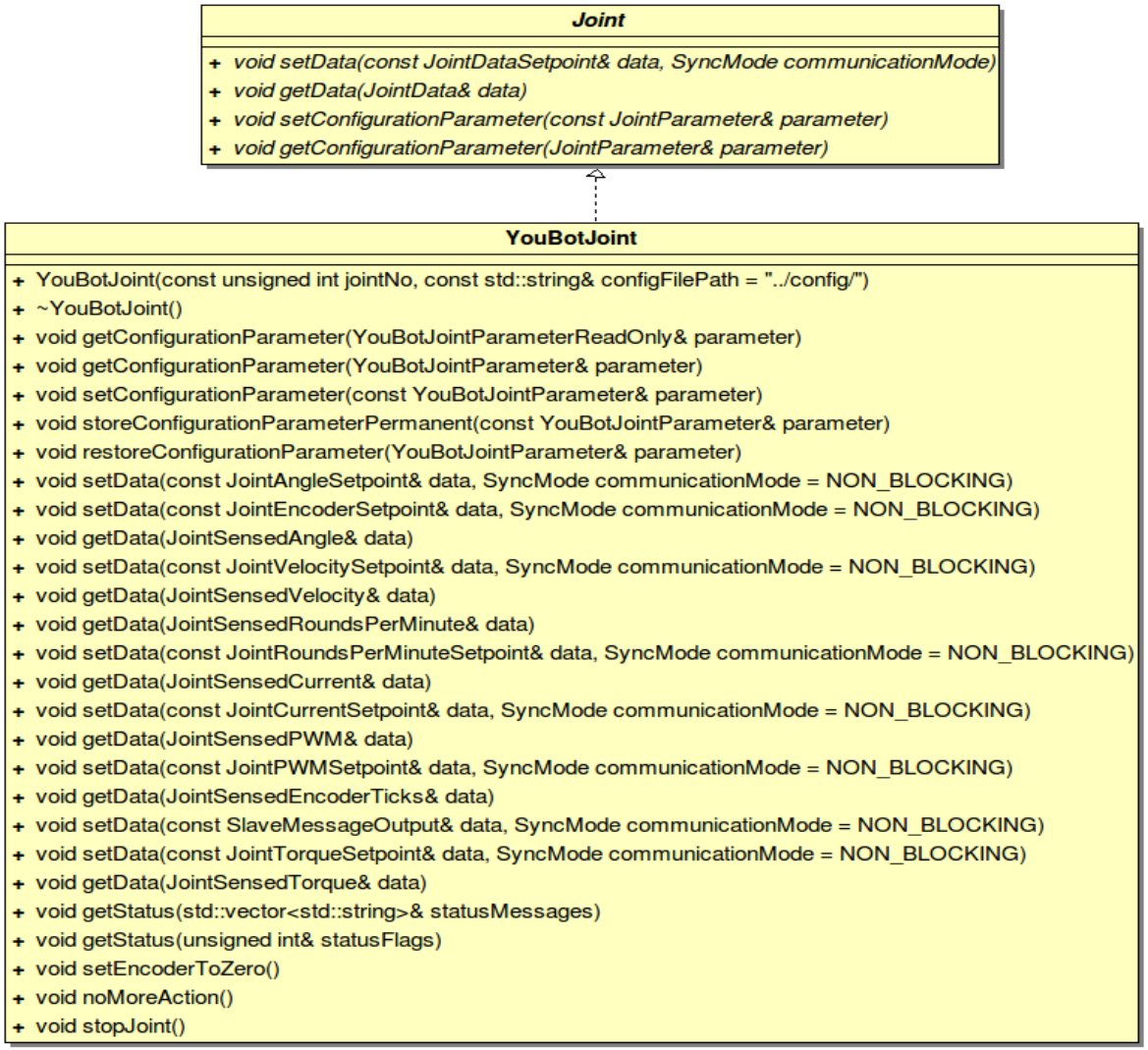
JointData Class Hierarchy



Parameter Class Hierarchy



YouBotJoint Class



YouBotBase and YouBotManipulator Class

YouBotManipulator

- int controllerType
- double minFirmwareVersion
- EthercatMaster& ethercatMaster

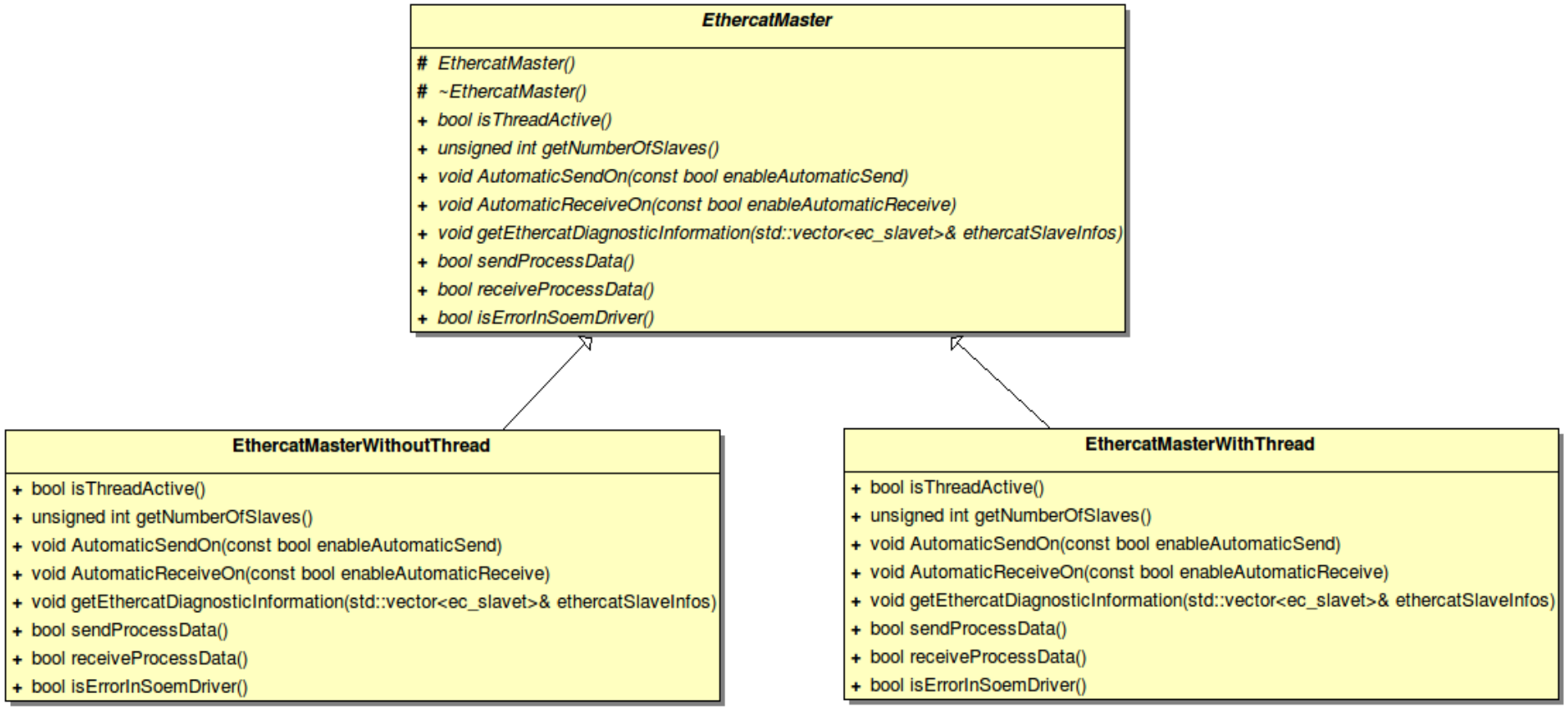
+ YouBotManipulator(const std::string name, const std::string configFilePath = "../config/")
+ ~YouBotManipulator()
+ void doJointCommutation()
+ void calibrateManipulator(const bool forceCalibration = false)
+ void calibrateGripper()
+ YouBotJoint& getArmJoint(const unsigned int armJointNumber)
+ YouBotGripper& getArmGripper()
+ void setJointData(const std::vector<JointAngleSetpoint>& JointData)
+ void getJointData(std::vector<JointSensedAngle>& data)
+ void setJointData(const std::vector<JointVelocitySetpoint>& JointData)
+ void getJointData(std::vector<JointSensedVelocity>& data)
+ void setJointData(const std::vector<JointCurrentSetpoint>& JointData)
+ void getJointData(std::vector<JointSensedCurrent>& data)
+ void setJointData(const std::vector<JointTorqueSetpoint>& JointData)
+ void getJointData(std::vector<JointSensedTorque>& data)
- bool areSame(const double A, const double B)

YouBotBase

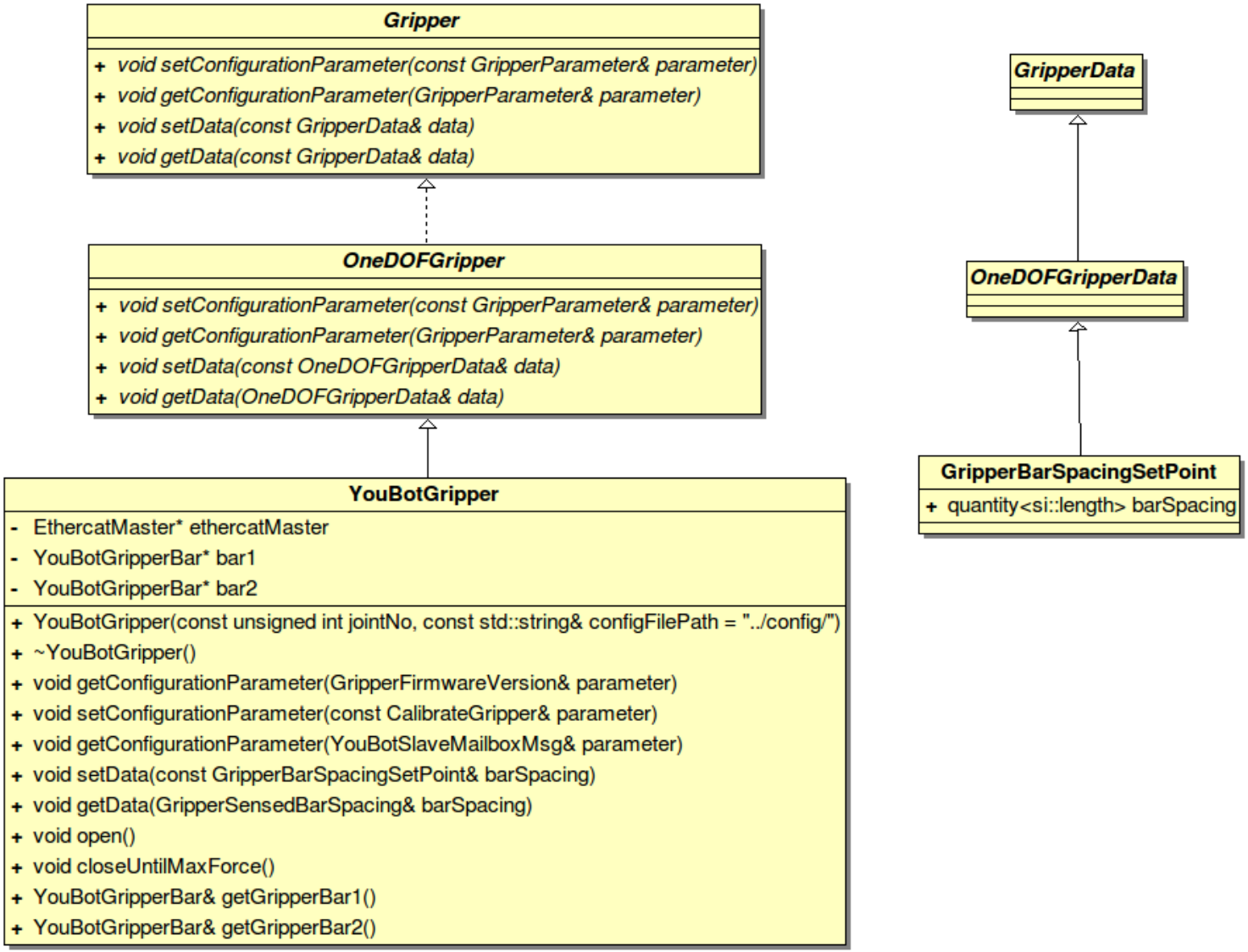
+ FourSwedishWheelOmniBaseKinematic youBotBaseKinematic
- int controllerType
- double minFirmwareVersion
- EthercatMaster& ethercatMaster

+ YouBotBase(const std::string name, const std::string configFilePath = "../config/")
+ ~YouBotBase()
+ YouBotJoint& getBaseJoint(const unsigned int baseJointNumber)
+ void getBasePosition(quantity<si::length>& longitudinalPosition, quantity<si::length>& transversalPosition, quantity<plane_angle>& orientation)
+ void setBasePosition(const quantity<si::length>& longitudinalPosition, const quantity<si::length>& transversalPosition, const quantity<plane_angle>& orientation)
+ void getBaseVelocity(quantity<si::velocity>& longitudinalVelocity, quantity<si::velocity>& transversalVelocity, quantity<si::angular_velocity>& angularVelocity)
+ void setBaseVelocity(const quantity<si::velocity>& longitudinalVelocity, const quantity<si::velocity>& transversalVelocity, const quantity<si::angular_velocity>& angularVelocity)
+ void setJointData(const std::vector<JointAngleSetpoint>& JointData)
+ void getJointData(std::vector<JointSensedAngle>& data)
+ void setJointData(const std::vector<JointVelocitySetpoint>& JointData)
+ void getJointData(std::vector<JointSensedVelocity>& data)
+ void setJointData(const std::vector<JointCurrentSetpoint>& JointData)
+ void getJointData(std::vector<JointSensedCurrent>& data)
+ void setJointData(const std::vector<JointTorqueSetpoint>& JointData)
+ void getJointData(std::vector<JointSensedTorque>& data)
- bool areSame(const double A, const double B)

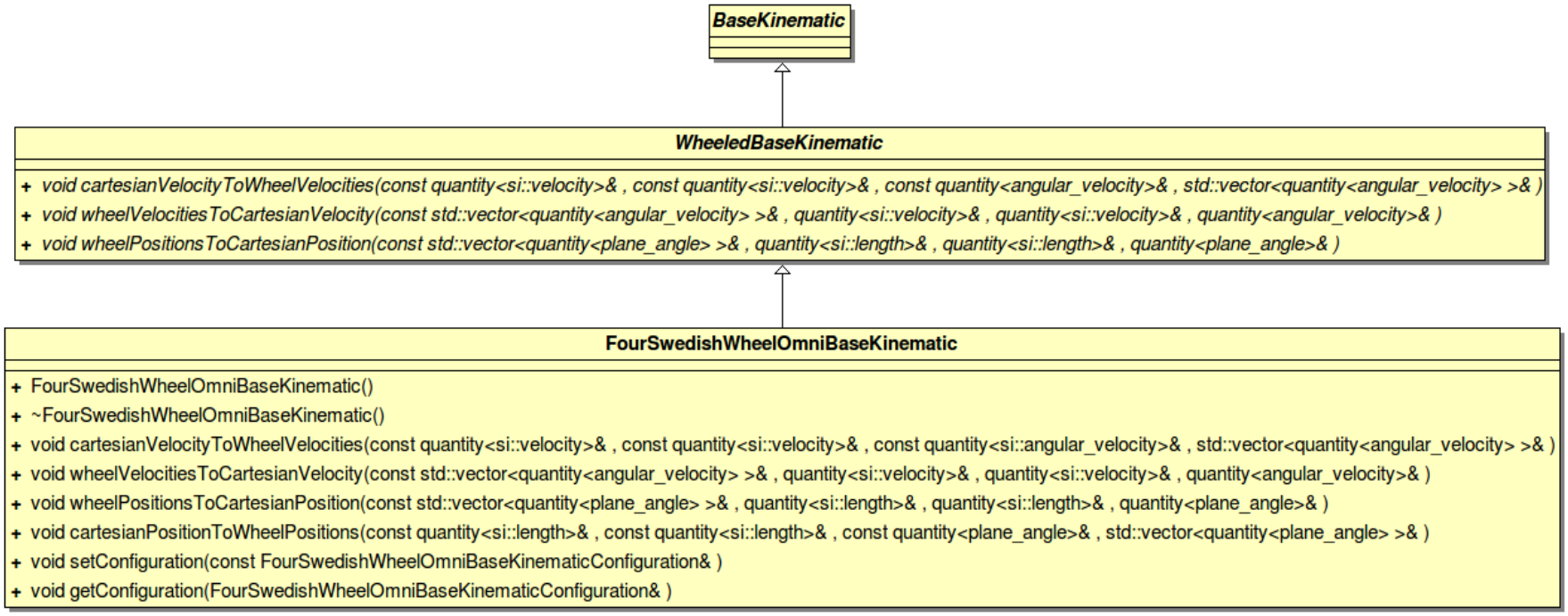
EtherCAT Master Class Hierarchy

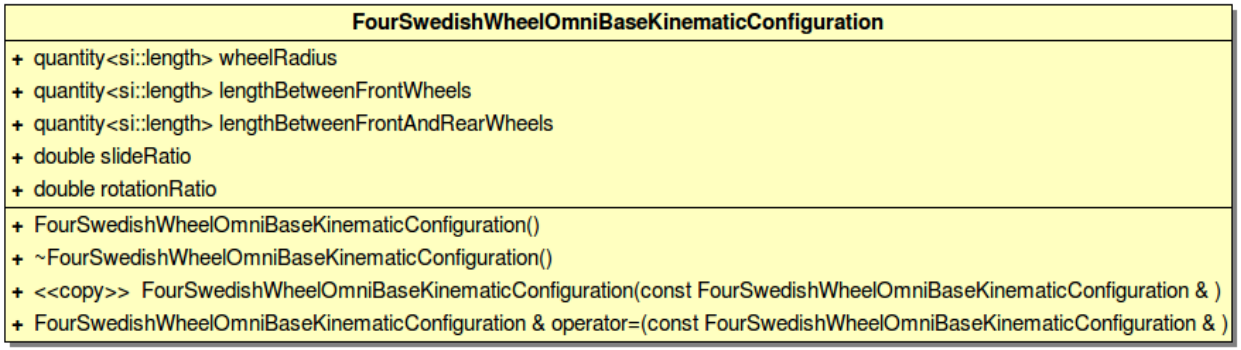


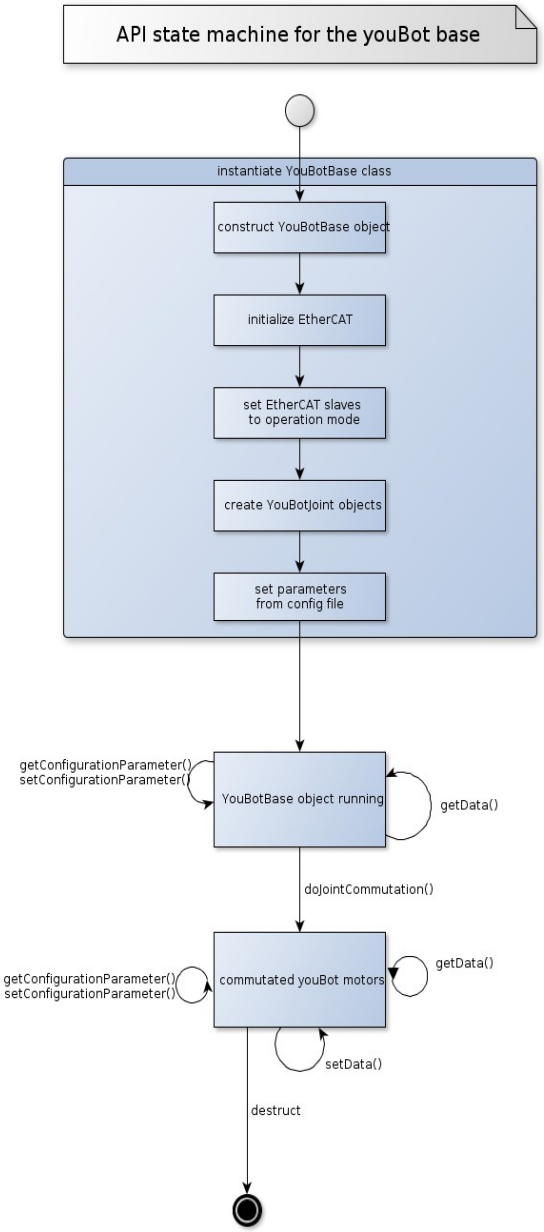
YouBotGripper Class Hierarchy

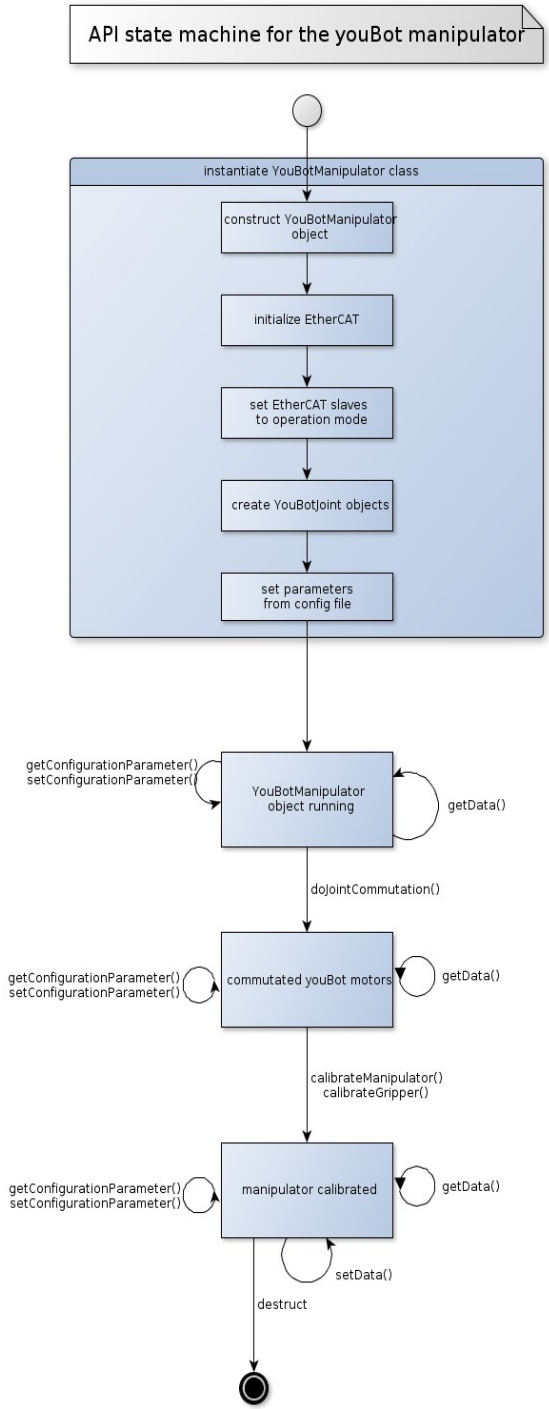


Base Kinematic Class Hierarchy






API state machine for the youBot base
instantiate YouBotBase class
construct YouBotBase object
initialize EtherCAT
set EtherCAT slaves to operation mode
create YouBotJoint objects
set parameters from config file
getConfigurationParameter()
setConfigurationParameter()
YouBotBase object running
getData()
doJointCommutation()
getConfigurationParameter()
setConfigurationParameter()
commutated youBot motors
getData()
setData()
destruct


API state machine for the youBot manipulator
instantiate YouBotManipulator class
construct YouBotManipulator object
initialize EtherCAT
set EtherCAT slaves to operation mode
create YouBotJoint objects
set parameters from config file
getConfigurationParameter()
setConfigurationParameter()
YouBotManipulator object running
getData()
doJointCommutation()
getConfigurationParameter()
setConfigurationParameter()
commutated youBot motors
getData()
calibrateManipulator()
calibrateGripper()
getConfigurationParameter()
setConfigurationParameter()
manipulator calibrated
getData()
setData()
destruct

API Configuration

The API configuration files are located in the config folder.

youbot-ethercat.cfg contains parameters for the EtherCAT communication:
- Ethernet device name
- cycle time of the communication thread
- ...

The topology of the youBot joint are in youbot-base.cfg, youbot-manipulator.cfg. You can assign to each YouBotJoint one EtherCAT slave.

Additionally the config files contain joint specific parameters like the gear ratio or the encoder ticks per round. These parameters are needed by the API to do proper calculations.

Base example

```
#include "youbot/YouBotBase.hpp"

using namespace youbot;

int main() {
  try {
    // creates a YouBotBase instance with the name "youbot-base"
// which is also the name of the config file
    YouBotBase myYouBotBase("youbot-base");

    // do the sine commutation of the base joints
    myYouBotBase.doJointCommutation();
```

Base example

```
// create variables to set and get the base cartesian velocity and pose
quantity<si::velocity> longitudinalVelocity = 0.2 * meter_per_second;
quantity<si::velocity> transversalVelocity = 0.0 * meter_per_second;
quantity<si::angular_velocity> angularVelocity = 0 * radian_per_second;
quantity<si::velocity> actualLongitudinalVelocity = 0 * meter_per_second;
quantity<si::velocity> actualTransversalVelocity = 0 * meter_per_second;
quantity<si::angular_velocity> actualAngularVelocity = 0 * radian_per_second;
quantity<si::length> actualLongitudinalPose = 0 * meter;
quantity<si::length> actualTransversalPose = 0 * meter;
quantity<si::plane_angle> actualAngle = 0 * radian;

// sets the base cartesian velocity
myYouBotBase.setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);

// reads the base cartesian velocity
myYouBotBase.getBaseVelocity(actualLongitudinalVelocity, actualTransversalVelocity, actualAngularVelocity);

// reads the base cartesian position which have been calculated from the odometry
myYouBotBase.getBasePosition(actualLongitudinalPose, actualTransversalPose, actualAngle);

// print the actual cartesian velocity
LOG(info) << "actual velocity longitudinal: " << actualLongitudinalVelocity <<
" transversal: " << actualTransversalVelocity << " angular: " << actualAngularVelocity;
```

Base example

```
  //command base joint 1 a velocity of 2 radian per second
  JointVelocitySetpoint setVel;
  setVel.angularVelocity = 2 * radian_per_second;
  myYouBotBase.getBaseJoint(1).setData(setVel);


} catch (std::exception& e) {
  std::cout << e.what() << std::endl;
} catch (...) {
  std::cout << "unhandled exception" << std::endl;
}
return 0;
}
```

Manipulator example

```
#include "youbot/YouBotManipulator.hpp"
using namespace youbot;
int main() {
  try {
    // creates a YouBotManipulator instance with the name "youbot-manipulator" which is also the name of the config file
    YouBotManipulator myYouBotManipulator("youbot-manipulator");
    // do the sine commutation of the arm joints
    myYouBotManipulator.doJointCommutation();
    // calibrate the reference position of the arm joints
    myYouBotManipulator.calibrateManipulator();

    //receive motor current form joint 1 of the manipulator
    JointSensedCurrent current;
    myYouBotManipulator.getArmJoint(1).getData(current);
    std::cout << "Current manipulator joint 1: " << current.current << std::endl;

    //read the maximum positioning velocity parameter from the manipulator joint 1
    MaximumPositioningVelocity maxPositioningVelocity;
    myYouBotManipulator.getArmJoint(1).getConfigurationParameter(maxPositioningVelocity);
    quantity<angular_velocity> velocity;
    maxPositioningVelocity.getParameter(velocity);
    std::cout << "Maximum positioning speed of joint 1: " << velocity << std::endl;

    //configure 2 radian_per_second as the maximum positioning velocity of the manipulator joint 1
    maxPositioningVelocity.setParameter(2 * radian_per_second);
    myYouBotManipulator.getArmJoint(1).setConfigurationParameter(maxPositioningVelocity);

  } catch (std::exception& e) { std::cout << e.what() << std::endl; }
  return 0;
}
```

Gripper example

```
#include "youbot/YouBotManipulator.hpp"
using namespace youbot;
int main() {
  Try {
    YouBotManipulator myYouBotManipulator("youbot-manipulator")
    // calibrate the reference position of the gripper
    myYouBotManipulator.calibrateGripper();

    // open the gripper 2 cm
    GripperBarSpacingSetPoint gripperSetPoint;
    gripperSetPoint.barSpacing = 0.02 * meter;
    myYouBotManipulator.getArmGripper().setData(gripperSetPoint);

  } catch (std::exception& e) {
    std::cout << e.what() << std::endl;
  } catch (...) {
    std::cout << "unhandled exception" << std::endl;
  }
  return 0;
}
```

Data trace

```
EthercatMaster* ethercatMaster = &EthercatMasterFactory::getInstance("youbot-ethercat.cfg", "../config/", true);
YouBotBase myBase("youbot-base");
int jointNO = 4;
DataTrace myTrace(myBase.getBaseJoint(jointNO), "Joint4VelocityTest");
JointVelocitySetpoint setVel;
setVel.angularVelocity = 0 * radian_per_second;
myBase.doJointCommutation();
myBase.getBaseJoint(jointNO).setEncoderToZero();
unsigned int startTimeStep1 = 1000, durationTimeStep1 = 3000, durationTimeStep2 = 3000, durationTimeStep3 = 2000;
unsigned int startTimeStep2 = startTimeStep1 + durationTimeStep1 + 2000;
unsigned int startTimeStep3 = startTimeStep2 + durationTimeStep2 + 2000;
unsigned int overallTime = startTimeStep3 + durationTimeStep3 + 1000;
quantity<angular_velocity> angularVelocity = 0 * radian_per_second;
myTrace.startTrace();
while (myTrace.getTimeDurationMilliSec() < overallTime) {
  if (myTrace.getTimeDurationMilliSec() > startTimeStep1 && myTrace.getTimeDurationMilliSec() < startTimeStep1 + durationTimeStep1)
    setVel.angularVelocity = 1 * radian_per_second;
  if (myTrace.getTimeDurationMilliSec() > startTimeStep1 + durationTimeStep1)
    setVel.angularVelocity = 0 * radian_per_second;
  if (myTrace.getTimeDurationMilliSec() > startTimeStep2 && myTrace.getTimeDurationMilliSec() < startTimeStep2 + durationTimeStep2)
    setVel.angularVelocity = 2 * radian_per_second;
  if (myTrace.getTimeDurationMilliSec() > startTimeStep2 + durationTimeStep2)
    setVel.angularVelocity = 0 * radian_per_second;

  if (myTrace.getTimeDurationMilliSec() > startTimeStep3 && myTrace.getTimeDurationMilliSec() < startTimeStep3 + (durationTimeStep3 / 2)) {
    angularVelocity = angularVelocity + 0.01 * radian_per_second;
    setVel.angularVelocity = angularVelocity;
  }
  if (myTrace.getTimeDurationMilliSec() >= startTimeStep3 + (durationTimeStep3 / 2) && myTrace.getTimeDurationMilliSec() < startTimeStep3 + durationTimeStep3) {
    angularVelocity = angularVelocity - 0.01 * radian_per_second;
    setVel.angularVelocity = angularVelocity;
  }
  if (myTrace.getTimeDurationMilliSec() > startTimeStep3 + durationTimeStep3)
    setVel.angularVelocity = 0 * radian_per_second;
  myBase.getBaseJoint(jointNO).setData(setVel);
  myTrace.updateTrace(setVel);
  SLEEP_MICROSEC(800);
}
myTrace.stopTrace();
myTrace.plotTrace();
```

Installation

1. Install a minimal installation of ROS. (see ros.org)

2. Clone the youBot API sources:

   git clone git://github.com/youbot/youbot_driver.git

3. Clone additional ros packages which include the SOEM (Simple Open EtherCAT master):

   git clone git://github.com/janpaulus/brics-external-packages-ros.git

4. Add both repository folders to the ROS_PACKAGE_PATH environment variable.

5. Compile the youBot driver by typing:

   rosmake youbot_driver --rosdep-install

Additional information

- youBot software available on https://github.com/youbot/youbot_driver
- Further information http://youbot-store.com