The Enhanced YouBot Controller

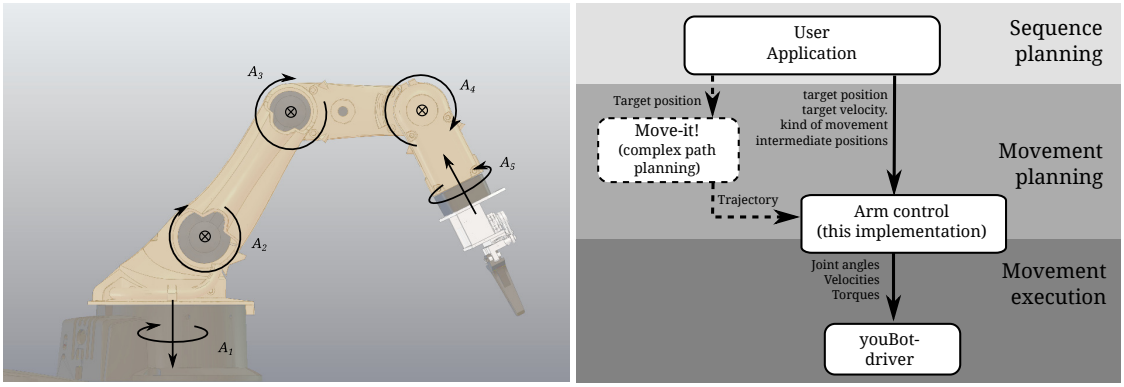
Simon Aden[1], Torben Carstensen[1], Jan Carstensen[1], Robin Kammel[1], Philipp Trunsheim[1], Andrej Dick[1], and Alexander Wentz[1]

LUHbots, Leibniz Universität Hannover
info@luhbots.de

Abstract The luh_youbot_os provides advanced control functionality for the KUKA youBot. It is a Robot Operating System (ROS) meta package consisting of controllers, tools and interfaces for the youBot. It has been used extensively during the RoboCup@Work competitions in 2015 by the LUHbots and is one of the core enhancements leading to the good performance. Three control concepts have been implemented for the manipulator: position, velocity and torque control. Graph based planning is used for complicated motions. The inverse kinematics are solved analytically. This approach can not be transferred to any manipulator, it is optimized for the youBot and can easily be configured for manipulators with the same set-up. Tests proved that all control methods can lead to a sub millimetre precision.

Keywords: mobile manipulation, youBot, controller

1 Introduction



(a) Axis of the youBot arm (b) Conceptual classification

Figure 1: Overview over the capabilities.

Complex manipulation tasks with mobile or stationary robots require arm control which allows different movement types and avoids collision during execution.
The preset software for the youBot only allows direct control of the manipulator's individual drives, which practically limits the arm's programming to sequences of predefined poses as joint angles.

The youbot_driver has been developed by the Bonn-Rhein-Sieg University of Applied Sciences. It consists of a C++ application programmable interface (API), which allows to communicate with the youBot drives via EtherCAT. The driver can only set or read drive values and does not offer sophisticated control.
The new arm control software replaces the ROS-Wrapper in its functionality to interface between the C++ API and ROS. It extends the functionality and offers action servers and topics for the aforementioned values. Furthermore it contains functionality for kinematics calculation, online and offline trajectory planning and trajectory execution, allowing for other tools e.g. MoveIt to be used.
This approach with adjustable length and size parameters is applicable to every robot with a five-degrees-of-freedoms manipulator and a configuration as illustrated in Figure 1a.

2 Kinematics

The basis for coordinated control of the manipulator are the kinematic relations. In the following the required kinematic equations will be introduced.

2.1 Cylindrical Coordinates

The end effector pose can well be described in cylindrical coordinates.

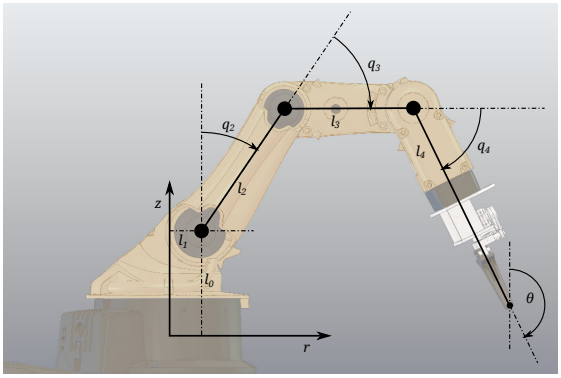


Figure 2: Geometry ot the youBot arm in the plane (Forward Kinematics)

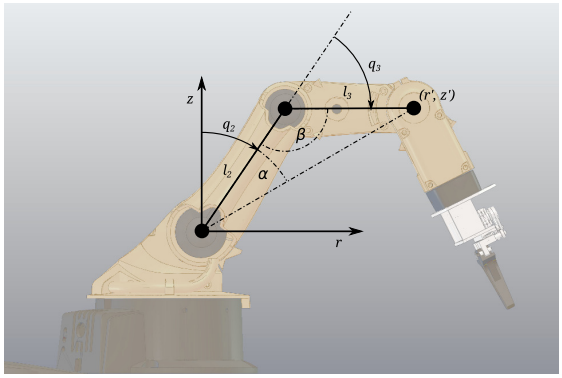


Figure 3: Reduced geometry of the youBot arm (Inverse Kinematics)

Forward Kinematics

$$\boldsymbol{q} = (q_1,\, q_2,\, q_3,\, q_4,\, q_5)^T \tag{1}$$

$$\boldsymbol{x}_E = (q_1,\, r,\, z,\, \theta,\, q_5)^T \tag{2}$$

$$r = l_1 + l_2 \sin(q_2) + l_3 \sin(q_2 + q_3) + l_4 \sin(q_2 + q_3 + q_4) \tag{3}$$

$$z = l_0 + l_2 \cos(q_2) + l_3 \cos(q_2 + q_3) + l_4 \cos(q_2 + q_3 + q_4) \tag{4}$$

$$\theta = q_2 + q_3 + q_4. \tag{5}$$

Inverse Kinematics The inverse kinematics can be calculated using the reduced geometry as shown in Figure 3:

$$r' = r - l_1 - l_4 \sin\theta \tag{6}$$
$$z' = z - l_0 - l_4 \cos\theta \tag{7}$$
$$\alpha = atan2\left(r', z'\right) \pm q_2 \tag{8}$$
$$\beta = \pi \pm q_3. \tag{9}$$

The sign in the addition in equation 8 and 9 is dependent on the configuration of the arm. In case of a raised elbow, it is negative, else positive. The same is true for the following equations.

The cosine rule for the triangle gives:

$$r'^2 + z'^2 = l_2^2 + l_3^2 - 2\, l_2\, l_3 \cos(\beta). \tag{10}$$
$$q_3 = \pi \pm \arccos\left(\frac{l_2^2 + l_3^2 - r'^2 - z'^2}{2\, l_2\, l_3}\right). \tag{11}$$
$$l_2^2 + r'^2 + z'^2 = l_3^2 + 2\sqrt{r'^2 + z'^2}\, l_2 \cos(\alpha), \tag{12}$$
$$q_2 = atan2(r', z') \pm \arccos\left(\frac{l_2^2 - l_3^2 + r'^2 + z'^2}{2\sqrt{r'^2 + z'^2}\, l_2}\right). \tag{13}$$
$$q_4 = \theta - q_2 - q_3. \tag{14}$$

Differential Kinematics For the differential kinematics we require the Jacobian matrix $J(q)$. We can derive it by partially differentiating the forward kinematics. Because q_1 and q_5 are not being transformed and are detached from the rest of the kinematics, the outer elements are equal to the identity matrix:

$$\boldsymbol{J}(\boldsymbol{q}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{\partial r(\boldsymbol{q})}{\partial q_2} & \frac{\partial r(\boldsymbol{q})}{\partial q_3} & \frac{\partial r(\boldsymbol{q})}{\partial q_4} & 0 \\ 0 & \frac{\partial z(\boldsymbol{q})}{\partial q_2} & \frac{\partial z(\boldsymbol{q})}{\partial q_3} & \frac{\partial z(\boldsymbol{q})}{\partial q_4} & 0 \\ 0 & \frac{\partial \theta(\boldsymbol{q})}{\partial q_2} & \frac{\partial \theta(\boldsymbol{q})}{\partial q_3} & \frac{\partial \theta(\boldsymbol{q})}{\partial q_4} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{15}$$

The inverse Jacobian matrix is calculated using its determinant and its adjugate:

$$\boldsymbol{J}^{-1}(\boldsymbol{q}) = \frac{1}{\det\left(\boldsymbol{J}\left(\boldsymbol{q}\right)\right)} adj\left(\boldsymbol{J}\left(\boldsymbol{q}\right)\right). \tag{16}$$

2.2 Cartesian Coordinates

A cylindrical end effector pose can be transformed to cylindrical coordinates with the following equations.

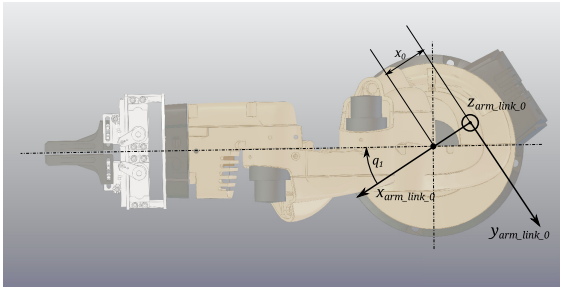


Figure 4: Cartesian coordinate system

$$x = r\cos q_1 + x_0 \tag{17}$$
$$y = -r\sin q_1 . \tag{18}$$

$$r = \sqrt{(x - x_0)^2 + y^2} \tag{19}$$
$$q_1 = -atan2\left(y, (x - x_0)\right). \tag{20}$$

3 Motion Types

The following motion types have been implemented.

3.1 Direct Goal Targeting

The direct goal targeting is the simplest form of arm control. Only the target position is specified. The appropriate joint angles are set directly as target values for the motor controls. Though the destination is reached very quickly, the uncoordinated movement of the joints results in an uncontrolled trajectory. In the simplest form the youBot driver already provides this kind of movement.

3.2 Interpolated Movement

The trajectory between a start point and a destination can be interpolated to gain better control of the movement. This enables e.g. precise motion of the end effector in a cartesian coordinate system or helps to avoid collisions by providing a predictable trajectory. Because the joint's acceleration is limited, a suitable velocity profile has to be chosen. For efficient online planning, a trapezoidal profile is used.

3.3 Graph-Based Planning

Interpolated movements are primarily suitable for small distances. Especially when interpolating in task space there is a chance of self collisions or reaching of joint limits. When dealing with movements over greater distances, a plan that only generates valid and collision free movements is recommended. For the construction of collision free trajectories a graph structure is used where the nodes are a set of predefined poses and the edges are motions between those poses that are known to be safe. A path along the edges of the graph therefore correlates with a sequence of safe arm movements.

3.4 Trajectory Presetting

As an interface to external trajectory planning algorithms there is the possibility to provide offline calculated trajectories. The external path planning can be realised with Move-it for example. Trajectories are defined by a sequence of joint angles and the associated points in time.

3.5 Velocity Control

Besides position commands it is possible to control the manipulator with velocities. This can be useful e.g. for synchronising the end effector position with a moving object. The velocity control mode uses the same online trajectory planning algorithm as interpolated position commands. Hence, a motion command in interpolation mode can be superimposed with a variable velocity command, allowing e.g. motion adjustment by visual servoing.

4 Interfaces

The motion types described in section 3 are implemented as a ROS-Node. This node acts as a central interface for controlling the robot and consists of multiple modules. Additional to the modules for different motion types, there are modules to control the mobile base and the gripper. The modular structure allows for extension in the future.

All motion commands are available as action servers in ROS. Additionally, a light-weight API in C++ is provided as a wrapper for the corresponding action clients to allow easy programming and to minimize code complexity in the client application. An example of the API is given in Listing 1.1.

```
#include <ros/ros.h>
#include <luh_youbot_controller_api/controller_api.h>

using namespace controller_api;

int main(int argc, char** argv)
{
// initialise ros
ros::init(argc, argv, "example_node");
ros::NodeHandle node_handle;

// initialise arm
YoubotArm arm;
arm.init(node_handle);

// move to predefined pose with default parameters
arm.moveToPose("ARM_UP");
arm.waitForCurrentAction();

// move to cartesian pose with Interpolation
luh_youbot_kinematics::CartesianPosition cart_pos;
cart_pos.setX(0.2);
cart_pos.setY(0.0);
cart_pos.setZ(0.05);
cart_pos.setTheta(M_PI);
```

```
cart_pos.setQ5(0.0);
arm.moveToPose(cart_pos, MotionMode::INTERPOLATE);
arm.waitForCurrentAction();

// move 5 cm forward
luh_youbot_kinematics::CylindricPosition relative_pos;
relative_pos.setR(0.05); // the rest is initialised with zeros

arm.moveToPose(relative_pos, MotionMode::INTERPOLATE, true);
arm.waitForCurrentAction();

return 0;
}
```

Listing 1.1: Example for using the programming gateway for the arm

4.1 Graphical User Interface

The GUI offers a direct and intuitive access to the motion commands. The interface can be used to test a particular movement or save predefined arm states.

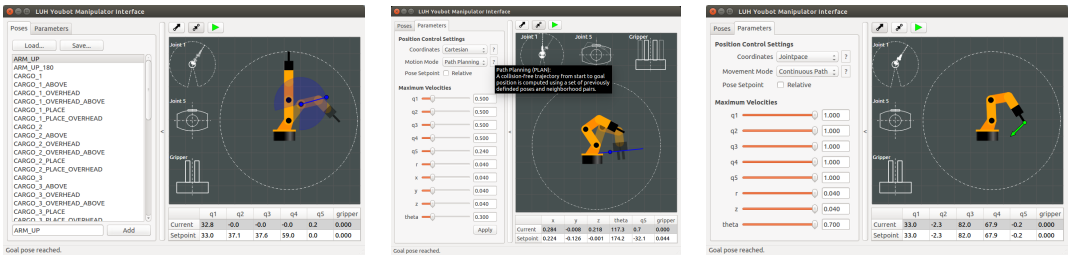


Figure 5: Graphical user interface - The interface allows for saving, and loading poses (left column) and changing the parameters of the manipulator.

On the left, predefined arm states can be loaded, adjusted and saved and parameters for the motion commands can be set. The right side shows a visualisation of the youBot arm. Both the current state and a variable target state are shown. The target position is either the chosen predefined arm state or a manually adjusted position. The latter can be achieved by clicking on the joints or the end effector and dragging them to the desired position with the mouse. Alternatively the desired target position can be set in a table on the lower right side. A velocity control is also possible. By dragging the joints or end effector in the velocity mode, velocity commands can be sent to the arm.

For easy teaching of new poses, a gravity compensation mode has been implemented. This allows an operator to move the actual arm into the desired position manually, therefore being very intuitive. It is possible to start and stop gravity compensation through the GUI.
Table 1 offers an overview into all packages provided.

Table 1: Generated and required ROS packages

Name	**Description**
luh_youbot_gui	Graphical user interface.
luh_youbot_driver_api	Wrapper library for abstraction of the youBot's driver functions. Based on the original ROS wrapper for the driver.
luh_youbot_joy_teleop	ROS node for controlling the youBot using a game controller or joystick. Both the mobile base and the arm can be controlled.
luh_youbot_kinematics	Kinematics library.
luh_youbot_controller	ROS node for motion execution with action server interface.
luh_youbot_controller_api	Programming Interface for motion commands.
luh_youbot_msgs	Contains all message, service and action definitions.
luh_youbot_poses	Contains a file with predefined arm states and a library for reading this file.
luh_youbot_description	URDF file for describing the youBot.
youbot_driver	The youBot driver with slight modifications (e.g. adapted for the build system catkin).
luh_laser_watchdog	Collision control for the base
luh_youbot_calibration	ROS node for calibrating the gravity compensation.
luh_youbot_gazebo	Wrapper library for abstraction of the youBot's driver functions for Gazebo.
luh_youbot_vrep_api	Wrapper library for abstraction of the youBot's driver functions for Vrep.

5 Conclusion and Outlook

The internal cascade controller implemented on the youBot's motor controllers allows the control via positions, velocities or electric currents. Given the motor constant and gear ratio the current control mode is equivalent to a torque control mode, because the motor current and torque are proportional. All three variations have been implemented as individual axis control in joint coordinates. To analyse the position deviation for the different control concepts, two exemplary test movements have been chosen. The first movement is an interpolated position command in joint coordinates and the second is a linear motion in cartesian coordinates. The joint angles have been measured during movement for evaluation. The specified end effector positions have not been measured directly, but have using forward kinematics from the joint angles.
The first measurements were done with firmware version 1.48 of the motor controllers and without modification of the default parameters. Because the results were not sufficient, the firmware was updated to version 2.00 and modifications of some motor controller parameters were tested. The best results (Figures (6-7) and Table (3)) have been reached with firmware 2.00 using position control mode and with deactivated internal ramp generator in the driver. The figures show the difference between planned and actual movement.

Table 3: Position deviation during movement in cartesian coordinates - the dynamic deviatation is the measured deviation during a movement, the static deviation is measured once the end position is reached.

			Control Mode		
			Position	Velocity	Torque
Joint angles	max. dynamic deviation	[°]	2.308	1.006	20.518
	max. static deviation	[°]	0.006	0.145	0.822
Position of the end-effector	max. dynamic deviation	[mm]	4.206	4.942	98.991
	max. dynamic deviation	[°]	0.069	0.383	19.662
	max. static deviation	[mm]	0.013	0.465	2.809
	max. static deviation	[°]	0.000	0.128	0.298

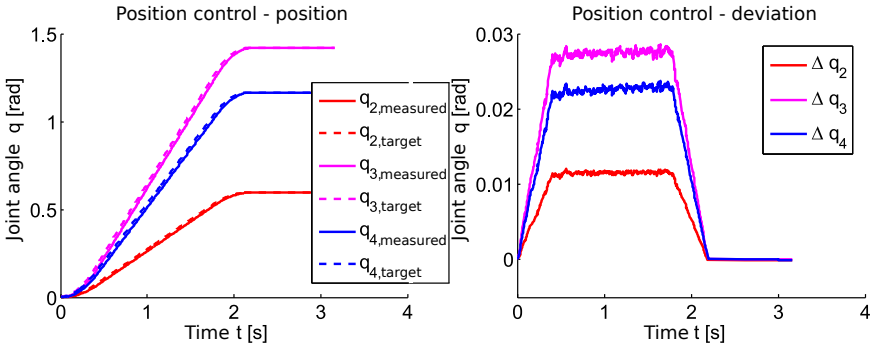


Figure 6: positional deviation jointspace (position control)

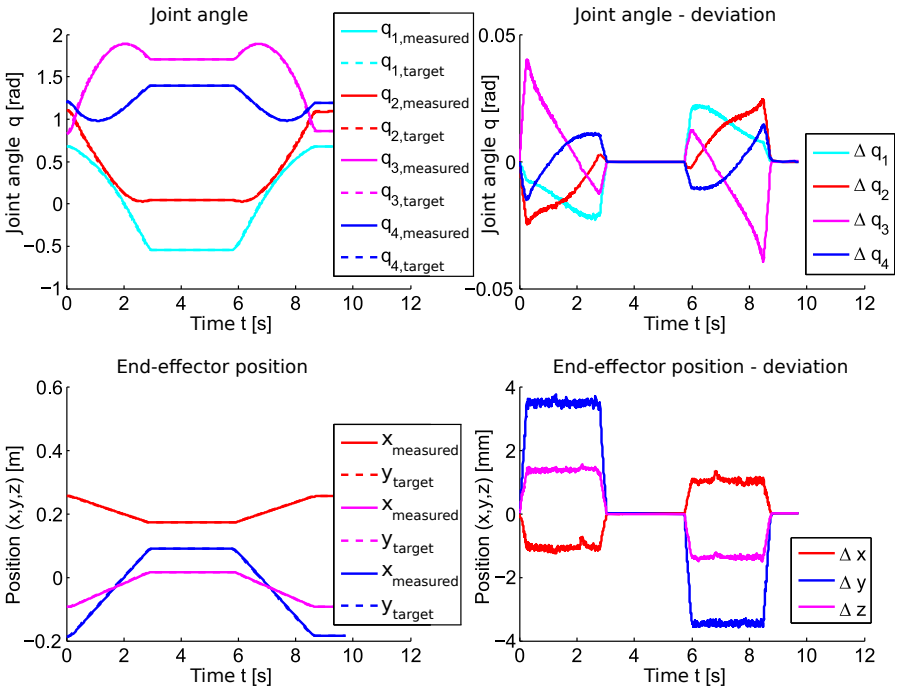


Figure 7: positional deviation cartesian (position control)