YARC - A Universal Kinematic Controller for Serial Robots Based on PMAC and MoveIt!

Yanyu Su* Yongzhuo Gao* Yan Wu† Wei Dong* Weidong Wang* Zhijiang Du*
*State Key Laboratory of Robotics and System, Harbin Institute of Technology, China
E-mail: suyanyu, gaoyongzhuo, dongwei, wangweidong, duzj01 at hit.edu.cn Tel: +86-451-86414271
†A*STAR Institute for Infocomm Research, Singapore
E-mail: wuy at i2r.a-star.edu.sg Tel: +65-64082719

***Abstract*—In this paper, we present a general approach to develop a kinematic controller for any serial robots using Programmable Multi-Axes Controller (PMAC) and MoveIt!. PMAC, a commercial product for motion control, is an all-in-one embedded system to control motion, amplify signals and acquire sensor information. MoveIt! is a state-of-the-art software for kinematics-based manipulations. It integrates many recent robotic advances, such as, Universal Robot Description File (URDF), Open Motion Planning Library (OPML) and so on. We will describe YARC (Yet Another Robot Controller), a principled way of integrating PMAC and MoveIt! together with a design of a smart control panel (TCPad). Several examples were demonstrated to show the functional superiority of this controller. The source code and hardware design have been made publicly accessible for the community.**

I. Introductions

Universal kinematic controller for robots has been studied for many years [1], [2], [3]. One challenge of developing such controller is the lack of general kinematic solvers and trajectory planners. As a result, many conventional universal kinematic controllers are usually limited to robots of particular configurations. Recently, many advances have been made to tackle this problem. One of these advances, MoveIt!, has been growing rapidly in the robotics community.

MoveIt! is a component of the Robot Operating System (ROS) [4]. By integrating Universal Robot Description File (URDF), Open Motion Planning Library (OPML) and other toolkits, MoveIt! is capable of solving kinematics, representing environments, planning trajectories with obstacle awareness and other useful functions for various kinematics-based applications. Although application of MoveIt! on various robotic platforms has been reported recently, porting it to the hardware of a new robotic platform still needs much unstructured manual work. To fill this gap, in this paper, we propose to implement an interface between MoveIt! and Programmable Multi-Axes Controller (PMAC), a commercial products for motion control.

PMAC is an all-in-one embedded system for motion control. It has been introduced to implement kinematic controller for many robotic platforms [5], [6], [7]. In some of the studies, PMAC is adopted as a multi-axis motion controller with the kinematics being solved on an Industrial Personal Comupter (IPC). Meanwhile, some studies also implement kinematics solver in PMAC since it provides an interface to solve for analytic kinematics. However, as analytic kinematics does not exist for many robots, this approach is not practical for developing universal kinematic controllers. In this paper, we adopt PMAC as a multi-axis motion controller and use MoveIt! to solve the kinematics and the planning problems.

The rest of the paper is organised as follows: We will present the technical background including ROS, MoveIt! and PMAC in Section II, followed by the design and the implementation of YARC, the proposed kinematic controller in Section III. We will present two real-life examples of applying the controller in Section IV.

II. Technical Backgrounds

A. ROS

ROS (Robot Operating System) is a meta-operating system with the philosophy to promote code reuse against the repetitive code work among different programmable robots. As an open-source programming framework, ROS is improved by hundreds of user-contributed ROS packages in the last few years [8], [9]. It provides standard operating system facilities such as hardware abstraction, low-level device control, implementation of commonly used functionalities. It is based on graph architecture with a centralised topology where processing takes place in nodes that may receive or post data, such as multiplex sensor data, control signals, states, planning, actuator values and so on. And data is passed between modules using inter-process communications.

B. MoveIt!

MoveIt! is a state-of-the-art software package for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. It provides an easy-to-use platform for developing advanced robotics applications, evaluating new robot designs and building integrated robotics products for industrial, commercial, research & development and other domains [10].

Given an URDF, MoveIt! can generate a kinematic model for a serial-chain manipulator, which many kinematic model-based algorithms could be implemented on. In this paper, we use the kinematic model generated by MoveIt! in motion planning algorithms for desired end-effector trajectories.

C. Programmable Multi-Axes Controller (PMAC)

PMAC is an embedded motion controller manufactured by Delta Tau. It has many attractive features for robot applications: 1) It provides control protocol through Ethernet; 2) It supports Coordinates System with coupled motion, which can be used for describing coupling designs in mechanical system(e.g. differential mechanism, parallel mechanism); 3) It implements the spline interpolation motion mode, which can be used to decrease the communication bandwidth without introducing unpredictable errors.

III. Design And Implementation of YARC

The goal in this work is to develop a kinematic controller (Yet Another Robot Controller, YARC) with a smart control panel (Teach & Control Pad, TCPad) design for serial robots based on ROS, MoveIT! and PMAC. The proposed controller should be able to carry out the tasks of point-to-point motions and trajectory motions in both joint space and Cartesian space. In the following sections, we will present the formulation of the tasks, followed by the design and the implementation of the proposed kinematic controller.

A. Problem Formulation

1) Kinematic Model: We describe the forward kinematics of a serial robot in the form

$$x = f(q), \tag{1}$$

where $x \in R^{n_x}$ denotes the position and orientation of an interested point on the robot, e.g. TCP (Tool Centre Point), $q \in R^{n_q}$ denotes the positions of the joints, $f(q)$ denotes the forward kinematic model of the robot. We also describe the inverse kinematics with

$$q = f^{-1}(x), \tag{2}$$

where $f^{-1}(x)$ denotes the inverse kinematic model of the robot.

2) Actuation Model: The position of the joints is related to the position of the actuators by

$$a = J_a * q + a_0, \tag{3}$$

where $a \in R^{n_a}$ denotes the positions of the actuators, $a_0 \in R^{n_a}$ denotes the offsets of the actuators (the actuator positions when the joints are at the home positions of the kinematic model), $J_a \in R^{n_q \times n_a}$ is the matrix describing the mapping from the joint positions to the actuator positions. If J_a is invertible, the positions of the actuators can be mapped to the positions of the joints with

$$q = J_a^{-1}(a - a_0), \tag{4}$$

where J_a^{-1} is the inverse matrix of J_a.

As the actuation model does not affect the planning in most motion control tasks, much work done in this area is based on the kinematic models. However, to apply these kinematic model-based advances onto a real robot, mapping of the abstracted kinematic model into a real mechanical structure using the actuation model has to be considered. Therefore, developing a controller that could simplify the mapping process would be practically helpful.

B. Overview

A straightforward approach to build kinematic controller for a robot system is to apply kinematics algorithms to generate joint trajectories for given tasks and use a multi-axis controller to execute the joint trajectories. Following this approach, we divide our kinematic controller into six functional modules: 1) a set of three kinematics algorithms for solving various tasks, 2) a multi-axis controller, 3) a graphic user interface, 4) a set of robot motion instructions and a TCPad for operating and programming the robot, 5) a coordinator that coordinates different modules, 6) a software framework that supports the communications between different modules.

Various toolkits are available for building these functional modules. In this work, we use a combination of ROS, MoveIt! and PMAC. The functional modules implemented in the proposed controller with corresponding toolkits are listed in Tab. I. However, as MoveIt! does not fully support motion planning for end-effector trajectory [1], which is essential for dexterous manipulations, we develop a module for this task, based on the kinematics library generated by MoveIt!. Furthermore, since the user-interface of MoveIt! is Rviz (a graphic user interface of ROS), which is not convenient for everyday use, we also develop a set of robot instructions and TCPad to enhance usability.

The software architecture of the controller is shown in Fig. 1. The controller consists of two parallel motion control pipelines. One is MoveIt! pipeline which consists of the user interface (Rviz) and the coordinator of MoveIt! (move_group). The other is the extensional pipeline which extends MoveIt! pipeline by providing a set of robot motion instructions, a TCPad and some extensional capabilities, e.g., master-slave motion control and motion planning for end-effector trajectories.

The module of interface sends commands and receives feedback from PMAC through the Ethernet protocol of PMAC. It is designed to coordinate with the helper (industrial_robot_client) for interfacing with MoveIt!.

The module of coordinator, which is working in the extensional pipeline, uses the kinematics provided by MoveIt! for solving the control tasks. With this design, any robot that can be modelled by MoveIt! could use this extensional pipeline.

As the kinematics provided by MoveIt! does not consider the actuation model, we implement it in PMAC using the coordinate system technique. With this design, all the kinematics-based algorithms can be implemented in MoveIt! pipeline or the extensional pipeline.

C. PMAC Settings

We implement the forward actuation model (3) as Coordinate System in PMAC, and implement the inverse actuation

[1] this function described as "Motion Planning (Desired end-effector trajectories)" in MoveIt! is experimental and not widely tested at the time when this paper published [10].

TABLE I
MAIN FEATURES AND CORRESPONDING TOOLKIT

Modules	Adopted Toolkit or Developed	Alternatives Toolkits
Algorithms for Solving Kinematics	MoveIt!	OpenRAVE, Simox, OMPL
Planning Algorithms with Obstacle Awareness	MoveIt!	OpenRAVE, Simox, OMPL
Planning Algorithms for End-Effector Trajectories	Developing	OpenRAVE, Simox, OMPL, MoveIt!
Multi-Axis Controller	PMAC	Elmo Mastro
Graphic User Interface	MoveIt!	OpenRAVE, Simox, Robotics Studio
Robot Motion Instructions and TCPad	Developing	-
Coordinator	MoveIt! & Developing	Microsoft Robotics Studio
Software Framework	ROS	Player, YARP, Microsoft Robotics Studio

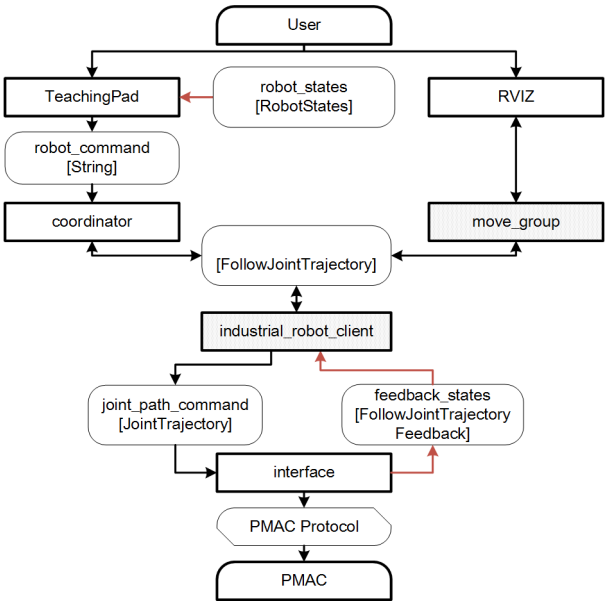


Fig. 1. Software Architecture of the controller. The user controls the robot through two pipelines, 1) the extensional pipeline (on the left) which provide a smart control panel, TCPad and 2) the MoveIT! pipeline (on the right) which provide a GUI module, RVIZ. Both pipeline uses a ROS "FoolowJointTrajectory" action to communicate with the industrial_robot_client, a helper for implementing interface to robot controller, which in this paper is the PMAC.

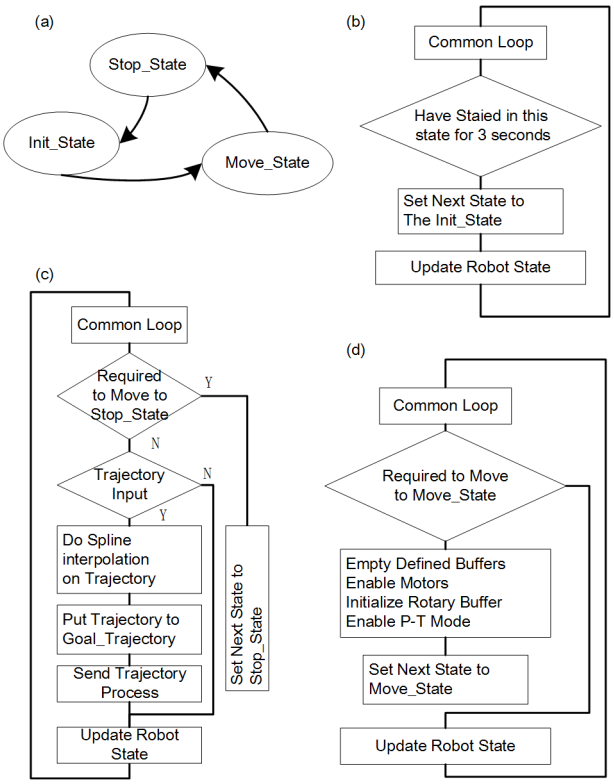


Fig. 2. The state machine of the interface is shown in (a). The work flow of Stop_State, Move_State and Init_State is respectively shown in (b) (c) and (d).

model (4) by updating predefined user defined variables in the foreground PLC program of PMAC as a example of PMAC commands shown in Code Listing 1.

```
!Define Coordinate System for joints
#1->25532.8278X
#2->25667.8750Y
#3->29308.7083Y + 29308.7083Z
#4->7110.1274A
#5->13285.5409B
#6->11104.5417C
!PLC Program to Calculate Feedbacks
OPEN PLC 0 CLEAR
P1=M162/25532.8278
P2=M262/25667.8750
P3=(M362+P2)/29308.7083
P4=M462/7110.1274
P5=M562/13285.5409
P6=M662/11104.5417
CLOSE
```

Listing 1. PMAC Settings Commands

D. The Interface

The interface is designed as a finite state machine, as shown in Fig. 2a. The programme initialises at the Stop-State in which a hard-stop command is sent to PMAC. After waiting for 3 seconds, it automatically moves to Init-State in which it initialises PMAC rotary buffer and clears all received trajectories before moving to the Move-State. In the Move-State, the programme receives, checks and interpolates the input trajectories before sending them to PMAC rotary buffer. Following the routine of the the helper (industrial_robot_client), the programme moves to the Stop-State, when the input of trajectory is empty.

1) Communicating with Motion Control Pipelines: The interface of MoveIt! to communicate with a multi-axis controller is the "FollowJointTrajectory" action. In order to make the interface compatible, we use the toolkit

of "industrial_robot_client" by subscribing to the topic "joint_path_command" of the "JointTrajecotry" messages and publishing "FollowJointTrajecotryFeedback" messages to the topic "feedback_states".

2) Communicating with PMAC: Delta Tau provides an open protocol based on TCP/IP protocol to access PMACs through Ethernet. We have developed an cross-platform open-source interface for PMAC with this protocol and adopted it in developing the interface.

3) Corresponding Motion Control Pipelines with PMAC: A joint in motion control pipelines (usually indexed by a name such as "joint_1") has to be mapped into PMAC (usually defined as a Coordinates System such as "x", and a variable for feedback such as "P1"). An example C++ code is shown in Code Listing 2.

```
/*Corresponding Axis from Pipelines to PMAC*/
/*                        Pipeline  C.S. F.B.*/
TRAJ_PROCESSOR.AddAxis ( "joint_1", "X", "P1" );
TRAJ_PROCESSOR.AddAxis ( "joint_2", "Y", "P2" );
TRAJ_PROCESSOR.AddAxis ( "joint_3", "Z", "P3" );
TRAJ_PROCESSOR.AddAxis ( "joint_4", "A", "P4" );
TRAJ_PROCESSOR.AddAxis ( "joint_5", "B", "P5" );
TRAJ_PROCESSOR.AddAxis ( "joint_6", "C", "P6" );
```

Listing 2. C++ Code for Corresponding Axis

4) Generating PMAC Commands of a Trajectory: In this work, we use the uniform cubic spline motion mode provided by PMAC. In this mode, each move on an axis of PMAC is computed as a cubic-splined position trajectory with uniform time interval. The intermediate positions in an interval are relaxed so as to avoid velocity or acceleration discontinuities.

5) Initialising PMAC: The process for initialising PMAC consists of four steps: 1) clear defined buffers in PMAC, 2) enable motors, 3) create rotary buffer, 4) initialising the uniform cubic spline motion mode. An example code is shown in Code Listing 3.

```
/*Clear Buffers*/
PMAC_OPERATOR.GetResponse("CLOSE ALL");
PMAC_OPERATOR.GetResponse("END GATHER");
PMAC_OPERATOR.GetResponse("DELETE GATHER");
PMAC_OPERATOR.GetResponse("DELETE ROT");
/*Enable Motors*/
PMAC_OPERATOR.GetResponse("#1J/#2J/#3J/#4J/#5J/#6J/" )
    ;
/*Define Rotary Buffer*/
PMAC_BUFFER.Create(1, p_RotaryBufferSize);
PMAC_BUFFER.Locate(0);
/*Initialize the Uniform Cubic Spline Motion Mode*/
PMAC_BUFFER.Open();
PMAC_BUFFER.WriteBuffer("CLEAR");
PMAC_BUFFER.WriteBuffer("SPLINE1 TM 50");
PMAC_BUFFER.Close();
```

Listing 3. Code for initialising PMAC

6) Sending Commands to the Rotary Buffer of PMAC: Following the instruction of using the rotary buffer of PMAC, we use the following three steps to send commands to the rotary buffer: 1) query the available space of the rotary buffer, 2) open rotary buffer, 3) sending some remaining lines of the trajectories that do not exceed the available space of the rotary buffer, 4) close rotary buffer.

E. The Coordinator

The coordinator is designed as a simple interpreter. It receives and checks robot commands to generate trajectories using corresponding kinematics-based algorithms. We define and implement a set of robot commands to fulfil the basic functional requirements of a TCPad, as shown in Tab. II.

From the examples of robot commands shown in Code Listing 4, the format of a typical command contains 7 parts: 1) the index of the command, 2) the name of the command, 3) the moving frame, 4) values, 5) motion time, 6) units and 7) the base frame.

```
P0001 MOVEL_RPY_LSPB {TCP} 0.00 1174.40 1176.00 0.00
    0.00 0.00 [20] (mm,deg,s) {B}
P0002 MOVEJ_SPLINE_LSPB {TCP} 0.00 0.00 0.00 0.00 0.00
     0.00; 30.00 15.00 30.00 -15.00 20.00 15.00; 0.00
     0.00 0.00 0.00 0.00 0.00 [20] (mm,deg,s) {B}
P0003 STOP
```

Listing 4. Examples of Robot Commands

F. TCPad

To provide seamless control experience for the users, we design and develop an open-sourced TCPad shown in Fig. 3. The core hardware of the TCPad is the ARM cortex-A8 with a 1GB RAM based on ARMv7 framework, which runs the Ubuntu 12.04 OS. The TCPad uses a 3-axis joystick with a button, a LCD touch screen and the membrane switches for user input, which improves the user interactive experiences. The mechanical structure of the TCPad is designed with consideration of ergonomic standards, which improves the operational flexibility. Meanwhile, it provides a structural support and protection for the electric system. Some basic functions are implemented on the TCPad, such as, control-teach-playback, calibration, coordinate system assignment, robot status display, workspace limit, system status display, user management, and system control. The TCPad uses the extensional pipeline to control a robot with the robot commands. The CAD, hardware specifications and the software of the TCPad are available for download [11].

G. Configuration for a New Robot

The following five steps should be carried out to adopt the presented controller for a new robot with a tutorial available online [11]:

- Build the kinematic model of the robot with URDF.
- Generate MoveIt! scripts using MoveIt! Setup Assistant.
- Configure PMAC according to actuation model.
- Modify corresponding settings in the Interface.
- Modify PMAC initialisation in the Interface.

IV. Implementation in Real-world Robots

The proposed controller was configured as the kinematic controller for two robots, a six DoF industrial robot and a tracked mobile robot with six Dof manipulator, as shown in Fig. 4. In this section, we describe the essential steps to configure the proposed controller for the two robots. The tutorial and demo videos are available online [11].

TABLE II
IMPLEMENTED ROBOT COMMANDS

Command Name	Function	Purpose For TCPad
STOP	stop current motion	general purpose
DMOVEJ	delta motion in joint space	controlling and teaching
DMOVE	delta motion in Cartesian space	controlling and teaching
MOVEJ	point-to-point motion in joint space	controlling
MOVEL	point-to-point motion with linear interpolation in Cartesian space	controlling
MOVEJ_SPLINE	multiple points motion with spline interpolation in joint space	controlling

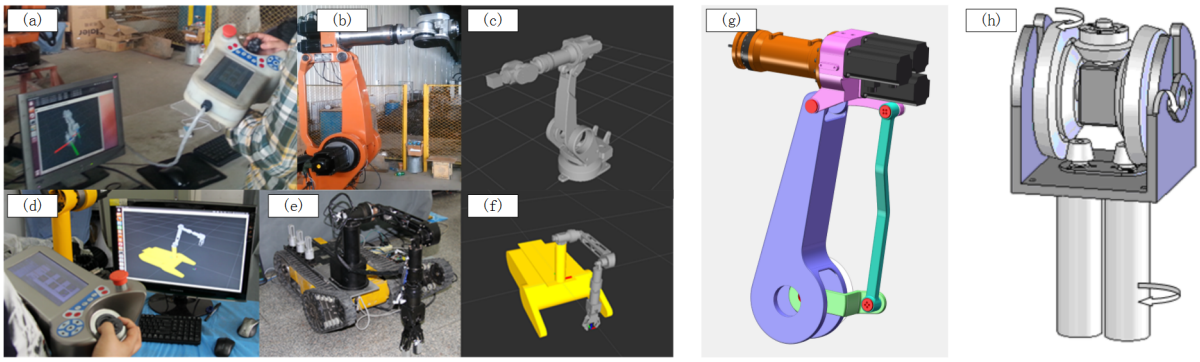


Fig. 4. Use Cases

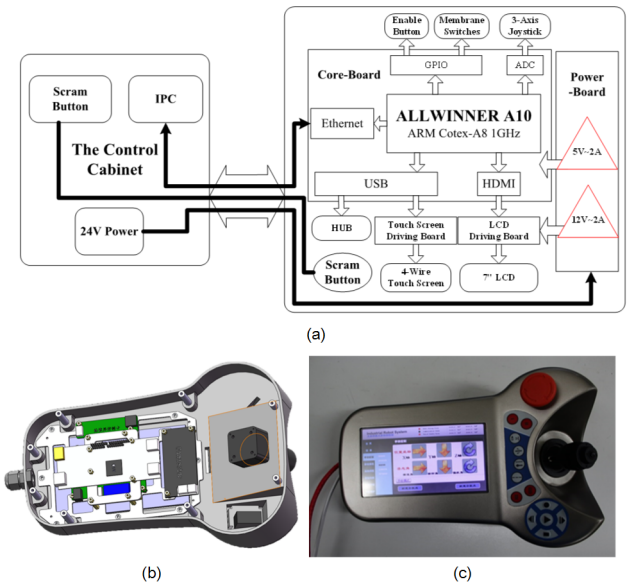
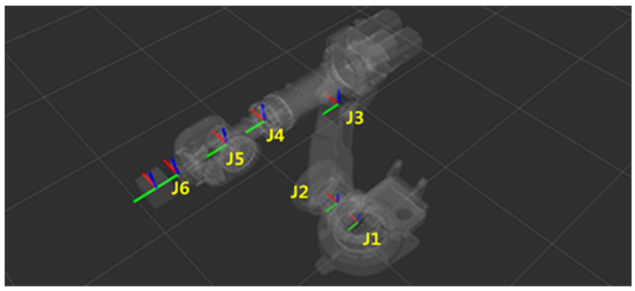


Fig. 3. The TCPad Design



	x	y	z	raw	pitch	yaw	ax	ay	az
J1->B	0	0	0.23	0	0	0	0	0	1
J2->J1	0	0.2	0.449	0	0	0	1	0	0
J3->J2	0	0	0.890	0	0	0	1	0	0
J4->J3	0	0.4369	0.195	0	0	0	0	-1	0
J5->J4	0	0.1995	0	0	0	0	1	0	0
J6->J5	0	0.238	0	0	0	0	0	-1	0

Fig. 5. Industrial Robot

A. Industrial Robot

1) Build Kinematic Model with URDF: It is not necessary to use D-H parameters to model the kinematic chain with URDF. As shown in Fig. 5, with the URDF, six parameters are used to fully describe the geometric relationship between any two joints with three further parameters to describe the orientation of the joint connecting them.

2) Generate MoveIt! Scripts: After writing the URDF file, use MoveIt! Setup Assistant to generate MoveIt! scripts by following the tutorial provided by MoveIt!.

3) Configure PMAC according to Actuation Model: A parallel mechanical transmission, as shown in Fig. 4g , is adopted in this industrial robot for better dynamic characteristics. This design makes the joint "J3" in the kinematic model couple with the joint "J2". We describe this with the actuation model

$$a = K * J_a * q + a_0, \tag{5}$$

where main diagonal matrix $K \in R^{6\times6}$ denotes the transmission ratio, the matrix J_a describes the coupled transmission

with

$$J_a = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (6)$$

PMAC Coordinate System definition, as shown in Code 1, can be written with the actuation model.

4) Modify corresponding settings in the Interface: The corresponding code has to be modified according to the names of each joint defined in URDF, and the Coordinate System name defined in PMAC. For example, the name of the first joint, as shown in Fig. 5, is defined as "J1" instead of "joint_1" defined in Line 3 of Code Listing 2, therefore, the code has to be changed as shown in Code Listing 5.

```
TRAJ_PROCESSOR.AddAxis ( "J1", "X", "P1" );
```

Listing 5. C++ Code for Corresponding Axis

5) Modify PMAC initialisation in the Interface: As the motors are respectively connected to PMAC port 1 to 6, there is no need to change PMAC initialisation code. But modification of Line 7 in Code 3 accordingly may be necessary if the hardware connection is different.

B. Mobile Manipulator

In this section, we illustrate how to change the controller configured for the industrial robot to a mobile manipulator.

1) Building Kinematic Model with URDF: We used an extension of Solidworks to build the URDF directly from the CAD model.

2) Generate MoveIt! Scripts: As the same as the example of industrial robot, we use MoveIt! Setup Assistant to generate MoveIt! scripts.

3) Configure PMAC according to Actuation Model: A differential mechanical transmission, as shown in Fig. 4h , is adopted in this robot. Similar to the industrial robot example, we can describe it with an actuation model

$$a = K * J_a * q + a_0, \quad (7)$$

where the matrix J_a is defined with

$$J_a = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0.5 & -0.5 \end{pmatrix}, \quad (8)$$

and change the PMAC Coordinate System definition respectively.

4) Other Steps: The other steps are the same as the example of industrial robot.

V. Conclusions

In this paper, we presented YARC, a universal controller for serial robot based on MoveIt! and PMAC. The design goal is to integrate MoveIt!, which is capable of solving kinematics and generating trajectories for manipulation tasks, and PMAC, which is capable of controlling various hardware. Three contributions are made in developing the controller: 1) We implemented an interface for PMAC according to the requirement of MoveIt!. 2) We implemented some features does not yet supported by MoveIt!, such as, planning for end-effector trajectory, interface for tele-operation and so on. 3) We also developed a TCPad for controlling and programming robots. The proposed controller has been applied to two robots as examples. The software and hardware of the proposed controller along with the source code of the examples are available online at http://bitbucket.org/suyanyucn/yarc/.

Looking forward, we envision several improvements that would increase the utility of the controller. The controller is currently adopting PMAC, which as a centralized motion controller requires many wire connections. We will investigate implement the controller using high-speed field bus based multi-axis motion controllers, such as, Elmo gold maestro which is based on EtherCAT.

References

[1] D. G. Bihn and T. C. S. Hsia, "Universal six-joint robot controller," *IEEE Control Systems Magazine*, vol. 8, no. 1, pp. 31–36, 1988.

[2] M. Naumann, K. Wegener, and R. D. Schraft, "Control Architecture for Robot Cells to Enable Plug'n'Produce," in *Proceedings IEEE International Conference on Robotics and Automation (ICRA 2007), Roma, Italy*, pp. 287–292, 2007.

[3] M. Sarabia, R. Ros, and Y. Demiris, "Towards an open-source social middleware for humanoid robots," *Proceedings IEEE-RAS International Conference on Humanoid Robots (Humanoids 2011), Bled, Slovenia*, pp. 670–675, 2011.

[4] S. Chitta, I. Sucan, and S. Cousins, "MoveIt!," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[5] D. Sun and J. Mills, "Torque control of sinusoidal PMAC motors for direct-drive robots," in *Proceedings 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pp. 53–58, IEEE, 2001.

[6] H. T. Tran, A. Khajepour, and K. Erkorkmaz, "Development and analysis of a PC-based object-oriented real-time robotics controller," in *Proceedings IEEE Conference on Control Applications (CCA 2005), Toronto, Ont., Canada*, pp. 1379–1384, 2005.

[7] H. Sang, J. Li, C. He, L. Zhang, and S. Wang, "Evaluations of a novel robotics assisted surgery system MicroHand A," in *Proceedings IEEE International Conference on Robotics and Biomimetics (ROBIO 2010), Tianjing, China*, pp. 1388 – 1392, 2010.

[8] S. Cousins, B. Gerkey, K. Conley, and W. Garage, "Sharing Software with ROS," *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 12–14, 2010.

[9] J. Boren and S. Cousins, "Exponential Growth of ROS," *IEEE Robotics & Automation Magazine*, vol. 18, no. 1, pp. 19–20, 2011.

[10] MoveIt!, Available online "http://moveit.ros.org/."

[11] Yanyu Su, Available online "http://www.suyanyu.com/wp/yarc/."