Coordinated jogging algorithm for general serial robot architectures

Martin Švejda * **Tomáš Myslivec** * **Martin Goubej** *

* *University of West Bohemia, NTIS Research Centre, Pilsen, Czech Republic (e-mail: msvejda@ntis.zcu.cz, tmyslive@ntis.zcu.cz , mgoubej@ntis.zcu.cz).*

Abstract: The paper deals with a novel functional block for general serial robot coordinate jogging. The functional block is implemented and integrated to the functional block library of the real-time control system REXYGEN and brings new possibilities of rapid prototyping and initial testing of advanced robot architectures. The robot kinematics is described by the Denavit-Hartenberg parameters. The common jogging mode of operation (world, tool) is supported as well as user-defined coordinate frames definition for specific applications. In the case of kinematic redundancy, the internal robot motion optimization is available allowing joint position limitation and/or obstacle avoidance. Maximum allowed joint velocity is taken into account in order to cope with the kinematic singularities. Illustrative examples of the early-prototype robot for industrial inspection purposes are introduced and universality of the proposed jogging functional block is verified.

Keywords: Industrial robot; rapid prototyping; coordinate jogging; REXYGEN control system; motion optimization; obstacle and limitation avoiding;

1. INTRODUCTION

Industrial robotics employs a large number of kinematic architectures ranging from simple ones like planar and SCARA robots or standard industrial arms/manipulators (serial 6 DoF robot with spherical wrist) to advanced and complex concepts like parallel or hybrid architectures, robots for special applications and experimental research prototypes. The robot controllers are often made to fit a specific robot type and it is supposed that the robot structure will not change except for standard setting options like tool and/or base compensation, etc. Especially in the case of research activities concerning new robot architectures development, it is necessary to have a possibility to test early robot prototypes in the manner of coordinate jogging in the base, tool or other specified coordinate systems (CS). In addition, the commercial controllers are closed systems where advanced and experimental features including e.g. redundancy resolution for optimal robot motion, robot kinematic reconfiguration or joint position/velocity/acceleration constraints can not been implemented in a simple way. In the current trend of rapid prototyping the tools for fast and efficient design of new robot architectures play crucial role in the field of robotics.

One of the possibilities to deal with the motion control of multi-axis mechatronic systems of non-standard user-defined kinematic architectures is well known Programmable Multi-Axis Controller (PMAC). PMACs are controls for which the entire hardware and software structure is organized around managing complex motion. The example of the PMAC is e.g. OMRON CK3E Series [12] with up to 32 controlled axes for advanced motion and CNC machining with G-code capability. The forward/inverse kinematic algorithm can be managed via matrix handling and the space conversion which makes possible e.g. complex kinematic control of hexapod telescope mirror.

Many software packages are devoted to multi-axis control of mechatronic systems including standard industrial robot configurations as well as user-defined general kinematic architectures. *Energid technologies corporation* offers highly sophisticated motion control for industrial, medical, commercial, collaborative and consumer robotic systems. Their *Actin SDK* [18] introduces "Real-Time Adaptive Motion Control software for Any Robot". *Actin SDK* offers possibility to define general kinematics and dynamics model, optimization of the motion (joint limit and singularity avoidance). *Energid Technologies Corporation* has been funded by NASA and *Actin 5 SDK* is labeled as software which brings enabling tools and capability for NASA space applications [3]. *MoveIt* motion planning framework [11] enables to incorporate latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. Over 100 robot kinematic architectures are included and users are allowed to write their own inverse kinematics algorithms (open/closed/branch kinematic loops) using *KLD* (numerical Jacobian-based solver: `http://www.orocos.org/kdl`). *IKFast Plugin* makes possible to implement own kinematics solvers. Some relevant applications are addressed to mobile manipulation task simulation [5], pick and place application with regards to given constraints (joints limit, obstacles, etc.) [7] and AUV control system implementation [21]. *RoboDK* [16] simulation environment brings offline robot programming features like optimization of robot path, avoiding singularities, axis limits and collisions overcoming. Moreover the trajectory post proces-

sor for a specific robot controller (ABB, FANUC, KUKA, MOTOMAN, etc.) is available in order to upload the program to native programming language. *RoboAnalyzer* [6; 15] is a 3D model based software that can be used to teach and learn the robotics concepts. It includes 3D model generator based on Denavit-Harteneberg parameters for serial manipulators with revolute and prismatic joints as well as functions for joint/cartesian level jogging, visualization of the robot movement, etc. *Virtual Robot Module* as part of the *RoboAnalyzer* can be integrated with Matlab.

Despite the range of advanced features provided by the above-mentioned tools, their integration into industrial grade environment remains difficult. Most of the generic PLC-based Motion Control (MC) systems rely on the PLCOpen Motion Control standard defining a functional block library implementing commonly required functionalities (PLCopen Motion Control, see `https://www.plcopen.org/technical-activities/motion-control`). The Coordinated Motion part of the standard covers both motion instructions (linear, circular and generic smooth path interpolation) and kinematic transforms. However, one of its main drawbacks is a lack of support for rapid prototyping of new robot architectures without time-consuming re-implementation of the forward/inverse kinematics whenever some change in the robot structure occurs. This includes also the functionality of the generic jog allowing to move the robot in a coordinated manner in a defined direction and a chosen coordinate reference system.

The goal of the paper is to propose an enhancement of the existing industrial MC blockset which implements a universal coordinated (Cartesian) robot jogging for a general serial robot kinematics with revolute joints. The key functionality requirements are given as follows:

- Arbitrary robot architectures with revolute joints (e.g. number of joints, rotation axis arrangement, etc.)
- Base or/and end-effector (tool) compensation
- End-effector (Cartesian) jogging including user-defined CSs reference (e.g. jog in specific plane only, etc.)
- Joints position limits detection and stopping the end-effector motion if the limits are reached
- Proximity to singularity detection and stopping the end-effector motion, further movement is allowed only in the directions going away from the singularity
- Limitation the maximum joint speed (no robot joints can rotate faster than given limit at the expense of slowing down the end-effector coordinated motion speed)
- Redundancy resolution (more joints than controlled end-effector DoFs) in the sense of joint position limit overcoming and/or user-defined objective function optimization

The proposed algorithm is implemented and experimentally validated by means of REXYGEN [17] real-time control environment and two examples of unconventional industrial robot designs.

2. THEORETICAL BACKGROUND

There are many notations for robot kinematic description, but the well known are probably Denavit-Hartenberg [4] and Khalil-Kleifinger [10] notations. In order to modelling only serial kinematic chains the best known Denavit-Hartenberg (D-H) notation is sufficient. The kinematics of the robot with n revolute joints is described by $3 \cdot n$ kinematic parameters d_i, a_i, α_i for $i = 1 \dots n$ and n joint coordinates θ_i. The meaning of the kinematic parameters is depicted in Fig 1. Compensation of the base ($\star = b$) and end-effector ($\star = e$) position are given by the parameters $x_\star$, $y_\star$, $z_\star$ (translation), $\gamma_\star$, $\beta_\star$, $\alpha_\star$ (ZYX Euler angles for orientation).

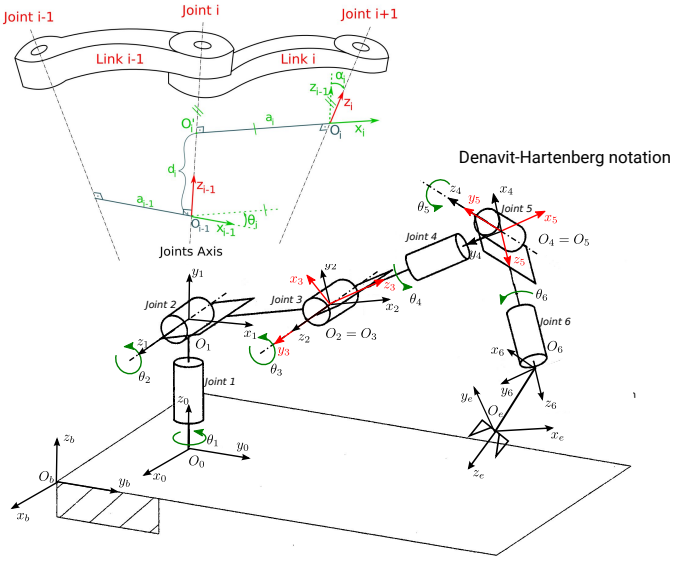

Fig. 1. Denavit-Hartenberg notation and its demonstration on 6 DoF serial industrial robot architecture

Note that the forward kinematics algorithm is given by successive multiplications of the homogeneous transformation matrices in the following form:

$$\boldsymbol{T}_e^b = \boldsymbol{T}_0^b \cdot \prod_{i=1}^{n} \boldsymbol{T}_i^{i-1} \cdot \boldsymbol{T}_e^n, \quad \boldsymbol{T}_i^{i-1} = \left[\begin{array}{c|c} \boldsymbol{R}_i^{i-1} & \boldsymbol{r}_{i-1,i}^{i-1} \\ \hline 0 \;\; 0 \;\; 0 & 1 \end{array}\right], \tag{1}$$

where $\boldsymbol{R}_i^{i-1}$ is rotation matrix and $\boldsymbol{r}_{i-1,i}^{i-1}$ is translation vector between two consecutive CSs, base resp. end-effector compensation are defined in a similar manner considering unique transformation from Euler angles to rotation matrix. A subscript indicates corresponding CS and a superscript indicates the reference CS. The homogeneous transformation matrices are defined according to D-H notation and they are depending on kinematic parameters and joint coordinates as follows:

$$\boldsymbol{T}_i^{i-1} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2}$$

2.1 Robot coordinated jogging problem

Robot coordinated jogging problem is defined as finding corresponding joint velocity for a given end-effector velocity with regards to a chosen CS (e.g. world, tool, etc.). The basic dependencies between joint and end-effector velocity are:

$$\dot{\boldsymbol{X}}_e^b = \boldsymbol{J}_e^b \cdot \dot{\boldsymbol{Q}}, \quad \dot{\boldsymbol{X}}_e^b = \begin{bmatrix} \dot{\boldsymbol{O}}_e^b \\ \boldsymbol{\omega}_e^b \end{bmatrix}, \tag{3}$$

where $\dot{\boldsymbol{O}}_e^b$ resp. $\boldsymbol{\omega}_e^b$ is translation resp. angular velocity of the end-effector CS with respect the base CS, $\dot{\boldsymbol{Q}}$ are joint velocity and $\boldsymbol{J}_e^b$ is kinematic Jacobian depending on joint position which can be generally and systematically computed from D-H kinematic parameters, see [13; 19].

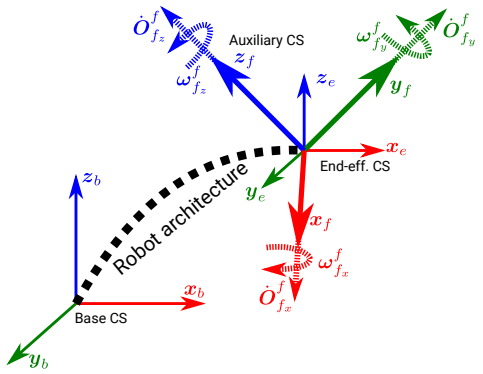

Fig. 2. Auxiliary end-effector CS definition

In order to reformulate the jogging of the robot end-effector CS F_e with respect to base CS F_b to general jogging scheme with respect to user-defined CS we define the auxiliary CS F_f, see Fig. 2 . The equation (3) results in:

$$\dot{\boldsymbol{X}}_f^b = \begin{bmatrix} {}^T\boldsymbol{R}_f^b & \boldsymbol{r2t}_f^b \\ \boldsymbol{t2r}_f^b & {}^R\boldsymbol{R}_f^b \end{bmatrix} \cdot \dot{\boldsymbol{X}}_f^f, \tag{4}$$

where

$$\dot{\boldsymbol{X}}_f^f = \begin{bmatrix} \dot{\boldsymbol{O}}_f^f \\ \boldsymbol{\omega}_f^f \end{bmatrix}, \quad \dot{\boldsymbol{X}}_f^b = \begin{bmatrix} \dot{\boldsymbol{O}}_f^b \\ \boldsymbol{\omega}_f^b \end{bmatrix}.$$

Because the new CS F_f is located in the origin of the end-effector CS F_e (CSs are only oriented to each other), the corresponding kinematic Jacobians are identical:

$$\boldsymbol{J}_f^b = \boldsymbol{J}_e^b. \tag{5}$$

Therefore it holds:

$$\begin{aligned} \dot{\boldsymbol{X}}_f^b &= \boldsymbol{J}_f^b \cdot \dot{\boldsymbol{Q}}, \\ \begin{bmatrix} {}^T\boldsymbol{R}_f^b & \boldsymbol{r2t}_f^b \\ \boldsymbol{t2r}_f^b & {}^R\boldsymbol{R}_f^b \end{bmatrix} \cdot \dot{\boldsymbol{X}}_f^f &= \boldsymbol{J}_f^b \cdot \dot{\boldsymbol{Q}}, \\ \dot{\boldsymbol{X}}_f^f &= \underbrace{\overbrace{\begin{bmatrix} {}^T\boldsymbol{R}_f^b & \boldsymbol{r2t}_f^b \\ \boldsymbol{t2r}_f^b & {}^R\boldsymbol{R}_f^b \end{bmatrix}^{-1}}^{\boldsymbol{P}} \cdot \boldsymbol{J}_f^b}_{\boldsymbol{J}_f^f} \cdot \dot{\boldsymbol{Q}}, \end{aligned} \tag{6}$$

$$\dot{\boldsymbol{Q}} = \left(\boldsymbol{J}_f^f\right)^{-1} \cdot \dot{\boldsymbol{X}}_f^f, \tag{7}$$

where $\dot{\boldsymbol{X}}_f^f$ is expected input of end-effector jogging velocity (velocity of the CS F_f with respect to the same CS), $\boldsymbol{J}_f^f$ is a new kinematic Jacobian and $\dot{\boldsymbol{Q}}$ is generated joint velocity. The rotation matrix ${}^T\boldsymbol{R}_f^b$ resp. ${}^R\boldsymbol{R}_f^b$ defines demanded end-effector jogging direction for translation resp. orientation and $\boldsymbol{r2t}_f^b$ resp. $\boldsymbol{t2r}_f^b$ are cross dependencies between demanded end-effector jogging direction for translation resp. orientation. The examples of common jogging mode can be summarized as follows:

- Word (base) jogging (jog in base CS)
$${}^T\boldsymbol{R}_f^b = {}^R\boldsymbol{R}_f^b = \boldsymbol{I}_{3\times3}, \quad \boldsymbol{r2t}_f^b = \boldsymbol{t2r}_f^b = \boldsymbol{0}_{3\times3}$$
- Tool (end-effector) jogging (jog in end-effector CS)
$${}^T\boldsymbol{R}_f^b = {}^R\boldsymbol{R}_f^b = \boldsymbol{R}_e^b, \quad \boldsymbol{r2t}_f^b = \boldsymbol{t2r}_f^b = \boldsymbol{0}_{3\times3}$$
where the rotation matrix $\boldsymbol{R}_e^b$ is the actual orientation of the end-effector CS.
- Word (base) translation jogging and tool (end-effector) orientation jogging
$${}^T\boldsymbol{R}_f^b = \boldsymbol{I}_{3\times3}, \quad {}^R\boldsymbol{R}_f^b = \boldsymbol{R}_e^b, \quad \boldsymbol{r2t}_f^b = \boldsymbol{t2r}_f^b = \boldsymbol{0}_{3\times3}$$

2.2 Redundancy resolution

The unique solution of the equation (6) exists only if the robot architecture is non-redundant. In the case that robot has more independent joints than DoFs of the end-effector, it is called redundant. Although kinematic redundancy is more often associated with the robots with more than six joints, the same case occurs for six axis robots if the motion of the end-effector is not fully controlled (e.g. only translation control for 6-axis industrial robot). While the number of controlled robot joints corresponds to the number of columns of the Jacobian, the number of controlled end-effector DoFs corresponds to the number of rows. The redundancy can be used for optimization of the robot (internal) motion resulting in maximization of defined joint position dependent objective function. Therefore, the optimal coordinate jog (6) is reformulated through additional term (which is projected to the the Jacobian null space) which generates internal robot joint velocity without the end-effector motion. For kinematic redundancy, the joint velocity is computed as follows, see [9; 13; 14]:

$$\dot{\boldsymbol{Q}} = \left(\boldsymbol{J}_f^f\right)^\dagger \cdot \boldsymbol{X}_f^f + \underbrace{\left(\boldsymbol{I} - \left(\boldsymbol{J}_f^f\right)^\dagger \cdot \boldsymbol{J}_f^f\right) \cdot \frac{\partial w}{\partial \boldsymbol{Q}}}_{\text{Optimization term}}, \tag{8}$$

where $\frac{\partial w}{\partial \boldsymbol{Q}}$ represents gradient of the joint dependent objective function $w(\boldsymbol{Q})$ to be maximized. The objective function is defined for the above mentioned key functionality requirements as follows:

- Minimization of the joint velocity (corresponding with singularity avoidance)
$$w(\boldsymbol{Q}) = 0. \tag{9}$$
- Joint position limit overcoming
$$\begin{aligned} w(\boldsymbol{Q}) &= -\frac{1}{2n} \sum_{i=1}^{n} \left(\frac{q_i - \bar{q}_i}{q_i^{\max} - q_i^{\min}} \right)^2, \\ \bar{q}_i &= q_i^{\min} + \frac{q_i^{\max} - q_i^{\min}}{2}, \end{aligned} \tag{10}$$
where $\boldsymbol{Q} = [q_1, q_2, \ldots, q_n]^T$ and $q_i^{\min}$, $q_i^{\max}$ define the joint position range of n robot actuators.
- Obstacles avoidance, see the example bellow.

2.3 Joint velocity limitation

The joint velocity limitation algorithm is based on the linear dependency (6) between joint and end-effector velocity. In the case of kinematic redundancy the additional joint velocity term resulting from optimization (8) belongs to the Jacobian null space and does not contribute to the end-effector velocity. Therefore the multiplication of the equation for the joint velocity (7, 8) by the constant factor

$$k = \max\left[1, \frac{jV_{\max}}{\max_{i=1\ldots n}\|q_i\|}\right], \quad \begin{array}{c}\dot{\boldsymbol{Q}} \to k\cdot\dot{\boldsymbol{Q}}\\ \dot{\boldsymbol{X}}_f^f \to k\cdot\dot{\boldsymbol{X}}_f^f\end{array}, \qquad (11)$$

results in limitation of the joint velocity to the allowed maximum value $jV_{\max}$. For $k < 1$ the end-effector motion remains in demanded (jogging) direction but it will be slowed-down. If the constant factor is less than the allowed minimum $k < k_{\min}$, (corresponding to maximum reduction of the end-effector speed) the robot motion is stopped. If the demanded jogging velocity $\dot{\boldsymbol{X}}_f^f$ is changed, the prediction to the next computation step (next time sample of the control executive) is computed and if the predicted factor $k > k_{\min}$ the robot motion is re-enabled. Therefore, the proposed algorithm allows the robot to get out of the singularity in any direction that moves away from it.

2.4 Joint position limitation

A simple algorithm is implemented which stops the robot motion ($k = 0$) ones of the joints enter to the position limit. The motion is re-enabled in the similar manner as above by comparing the actual and predicted robot joint position.

3. IMPLEMENTATION OF THE NEW GENERAL JOGGING BLOCK TO THE REXYGEN LIBRARY

REXYGEN real-time control system, see [17], is based on the programming without hand coding using the libraries of the functional blocks (e.g. similar to Matlab/Simuling programming). The general jogging block generates the demanded joint velocity depending on the current robot state (joint position) and demanded end-effector velocity (jogging). Implementation of the new functional block `JOG`, see Fig. 3, is based on the user programmable block `REXLANG` (`https://www.rexygen.com/doc/ENGLISH/MANUALS/BRef/REXLANG.html`) which implements an user-defined algorithm written in a scripting language very similar to the C language (or Java). All the above mentioned algorithms (Jacobian $\boldsymbol{J}_e^b$ and user defined-transformation matrix $\boldsymbol{P}$ computation, redundancy resolution and joint velocity and position limitation) are implemented via the REXLANG functional block. The inputs, outputs and configuration parameters are summarized as follows:

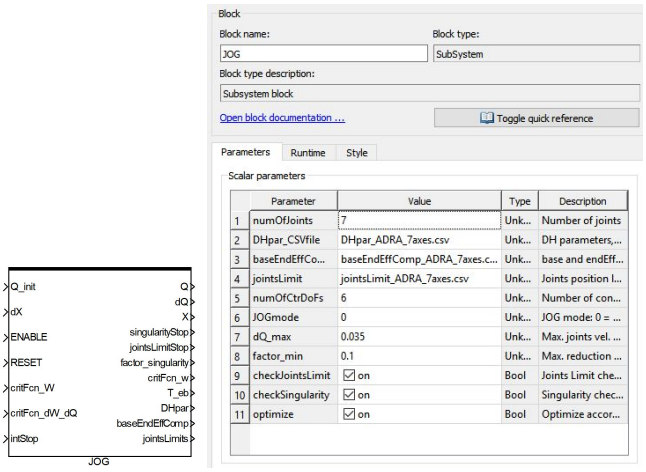

Fig. 3. General jogging functional block

The inputs, outputs and configuration parameters are summarized as follows:

Inputs: `Q_init` initial position of the joints for integration; `dX` end-effector jogging velocity; `ENABLE` enable block; `RESET` reset block internal variables; `critFcn_W` resp. `critFcn_dW_dQ` value of objective function w resp. its gradient $\frac{\partial w}{\partial \boldsymbol{Q}}$ (computed in other task, see bellow); `intStop` stop integrating (for testing purposes only);

Outputs: `Q` resp. `dQ` demanded robot joint position resp. velocity; `X` actual robot end-effector position; `singularity Stop` if true robot enters the nearly singularity position; `jointsLimitStop` nonzero value indicates corresponding joint position limitation; `factor_singularity` is factor k; `T_eb` is matrix $\boldsymbol{T}_e^b$; `DHpar` D-H parameters table; `baseEndEffComp` base and end-effector compensation table; `jointsLimits` joint position limitation table;

Parameters: `numOfJoints` number of robot joints; `DHpar_CSVfile` D-H parameters cfg file name; `baseEndEffComp_CSVfile` base, end-effector compensation cfg file name; `jointsLimit` joint position limit cfg file; `numOfCtrDoFs` number of controlled end-effector DoFs (number of accepted rows of jacobian $\boldsymbol{J}_f^f$); `JOGmode` mode of the end-effector jogging (world, tool or user-specified); `dQ_max` maximum joint speed; `factor_min` threshold $k_{\min}$, max. end-effector speed reduction; `checkJointsLimit` enable joint position limit checking; `checkSingularity` enable singularity checking; `optimize` enable optimization according to external defined objective function (else $w = 0$);

For `optimize = true` the external objective function is accepted and it is computed in separate task (typically with slow sample time) via the functional block `critFcn`, see Fig. 4. The block is implemented for two types of motion optimization: 1) Joint position limit overcoming, 2) Obstacle avoidance (obstacles are defined as cylinder surfaces with given axis, diameter, an infinite heigh and weight factors of the given obstacle for each robot link to avoid the obstacle).

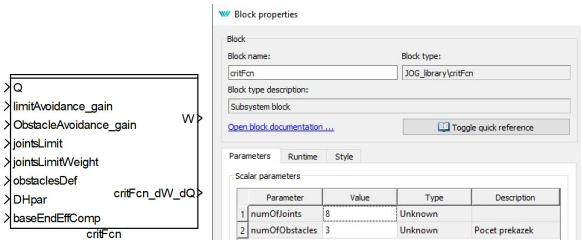

Fig. 4. Objective function computation functional block

The inputs, outputs and parameters are as follows (without those mentioned above):

Inputs: `limitAvoidance_gain` resp. `ObstacleAvoidance_gain` adjust the trade-off between the optimization modes; `jointsLimitWeight` defines the weights for joint position limit overcoming; `obstacleDef` definition of the obstacles parameters (see above);

Parameters: `fileName` user-defined `REXLANG` source code for objective function computation; `numOfObstacles` number of cylinder obstacles;

4. CONCLUSION AND ILLUSTRATIVE EXAMPLES

Two inspection robot prototypes ROBIN 1 and ROBIN 2 (ROBotic INspection) following the inspection robot research summarized in [8] are considered as illustrative example of benefits of the new `JOG` functional block for rapid prototyping of the non-standard robot architectures. The prototyping process of the robot includes a number of modifications in the architecture (number and geometry of the joints/links, specific jogging mode, etc.) where the basic robot coordinate motion properties (jogging) has to be tested every time.

4.1 ROBIN 1

The ROBIN 1 robot is a modular tiny robot for Non Destructive Testing applications where each module consists of two mutually perpendicular revolute axes (actuated by spindle drives or belt driven actuators alternatively [1]), see Fig. 5. We consider three modules which results in 6-axis robot. The main aim of the robot is positioning of the laser pointer spot (LPS) on defined projection plane given by the point $\boldsymbol{O}_r$, normal vector $\boldsymbol{z}_r$ and direction $\boldsymbol{x}_r$. The position of the LPS on the plane is dependent on translation and orientation of the robot end-effector. Regarding dependency (6) we suppose substitution $\dot{\boldsymbol{X}}_f^f = \dot{\boldsymbol{X}}_{las}^r$ (translation velocity in XY plane with respect to plane CS) and after some computations it can be shown that the user-defined matrix $\boldsymbol{P}$ for coordinate jogging in the user defined CS will have the following form:

$$\boldsymbol{P} = \begin{bmatrix} {}^T\boldsymbol{R}_f^b & \boldsymbol{r2t}_f^b \\ \boldsymbol{t2r}_f^b & {}^R\boldsymbol{R}_f^b \end{bmatrix} = \begin{bmatrix} \left(\boldsymbol{R}_r^b\right)^T \cdot \boldsymbol{N} \\ \boldsymbol{O}_3 \end{bmatrix},$$

where $\boldsymbol{R}_r^b = [\boldsymbol{x}_r^b\, \boldsymbol{y}_r^b\, \boldsymbol{z}_r^b]$ is the known orientation of the projection plane and

$$\boldsymbol{N} = \boldsymbol{z}_e^b \cdot \boldsymbol{M} + \left[\boldsymbol{I}_{3\times3} | - K \cdot \boldsymbol{S}(\boldsymbol{z}_e^b)\right],$$

$$\boldsymbol{M} = \frac{1}{\left(\boldsymbol{z}_r^{bT}\boldsymbol{z}_e^b\right)^2} \cdot \left[-\boldsymbol{z}_r^{bT} \cdot \left(\boldsymbol{z}_r^{bT}\boldsymbol{z}_e^b\right) | - \boldsymbol{z}_r^{bT}\left(\boldsymbol{O}_r^b - \boldsymbol{O}_e^b\right)\boldsymbol{z}_r^{bT}\right] \cdot \begin{bmatrix} \boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & -\boldsymbol{S}(\boldsymbol{z}_e^b) \end{bmatrix},$$

$$K = \frac{\boldsymbol{z}_r^{bT} \cdot \left(\boldsymbol{O}_r^b - \boldsymbol{O}_e^b\right)}{\boldsymbol{z}_r^{bT}\boldsymbol{z}_e^{bT}},$$

where $\boldsymbol{S}(\star)$ denotes skew-matrix generated by the vector and $\boldsymbol{O}_e^b$ resp. $\boldsymbol{z}_e^b$ are the translation resp. z-axis of orientation of the end-effector given by homogeneous transformation matrix $\boldsymbol{T}_e^b$. Only the joint position limit overcoming was used to resolve the redundancy of the robot (six controlled actuator axes and only two controlled end-effector DoFs). That makes possible to handle a large workspace despite the joints position range is strictly limited due to mechanical construction of the robot actuators.

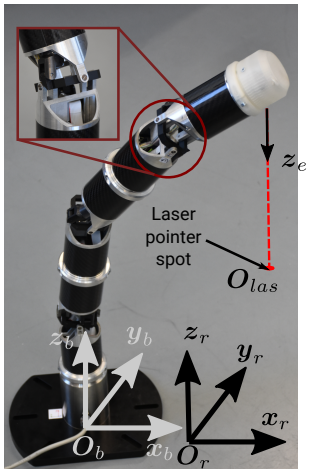

Fig. 5. ROBIN 1

4.2 ROBIN 2

The ROBIN 2 robot is a prototype of the experimental robot for inspection of pipeline welds in highly restricted areas, see Fig. 6. It consists of 1 DoF circumferential travel, 6 DoF serial chain and 1 DoF orientating (heading) of the test probe. The test probe is mounted via universal joint and it is supposed to be moved on the pipe surface with prescribed heading which results in 4 controlled DoFs of the end-effector (translation along and around the pipe and away from the pipe surface and rotation of the probe). The user-defined matrix $\boldsymbol{P}$ for coordinate jogging on the

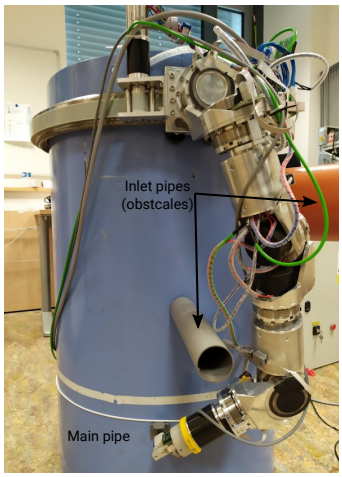

Fig. 6. ROBIN 2

pipe is simply defined by the transformation of Cartesian end-effector coordinates to cylindrical coordinates and its derivation is not further shown for simplification. The main challenge is to cope with the obstacle avoidance where the obstacles are supposed to be so called inlet pipes which enter the main pipe. Therefore, the obstacles can be described as a cylinder surface of given axis, center point and diameter. In order to implement the objective function inside the functional block `critFcn`, the obstacles will be projected to a so-called developed view, see Fig. 7. Mathematical background of the objective function computation is based on the formulation of minimum distance of the robot links and their extremal points (typically joint centers) from the cylinders axes.

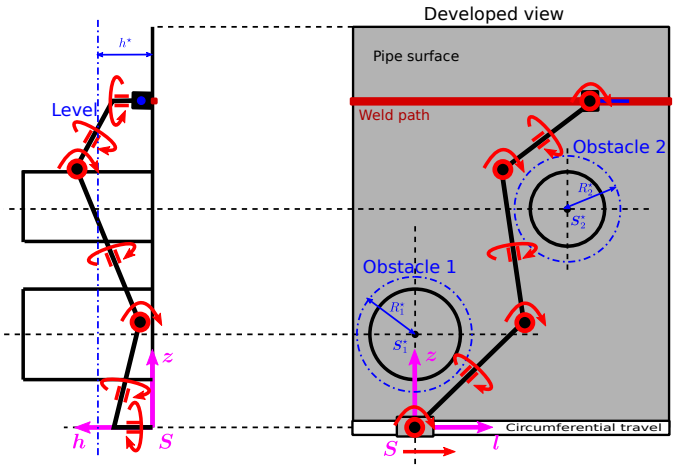

Fig. 7. Obstacles definition in the developed view

More information is given in [20; 2]. Demonstration of the end-effector jogging with obstacle avoidance is available at: `https://drive.google.com/file/d/1-fn98adnHwpLPRQWruKe9vBh850-H3r/view?usp=sharing`

ACKNOWLEDGEMENTS

This work was supported by the Ministry of Industry and Trade under Grant No. FV10044.

REFERENCES

[1] L. Bláha. Design of a compact 2-DoF joint with belt driven actuators. In *2017 IEEE 4th International Conference on Soft Computing Machine Intelligence (ISCMI)*, pages 198–202, Nov 2017. doi: 10.1109/ISCMI.2017.8279626.

[2] L. Bláha and M. Švejda. Path planning of hyper-redundant manipulator in developed view. In *2018 19th International Carpathian Control Conference (ICCC)*, pages 295–300, May 2018. doi: 10.1109/CarpathianCC.2018.8399644.

[3] D. Comstock, D. Lockney, and C. Glass. A structure for capturing quantitative benefits from the transfer of space and aeronautics technology. In *International Astronautical Congress, Cape Town, South Africa*, October 2010.

[4] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. of the ASME. Journal of Applied Mechanics*, 22:215–221, 1955. URL `http://ci.nii.ac.jp/naid/10008019314/en/`.

[5] H. Deng, J. Xiong, and Z. Xia. Mobile manipulation task simulation using ros with moveit. In *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 612–616, July 2017. doi: 10.1109/RCAR.2017.8311930.

[6] V. Gupta, R. Chittawadigi, and K. Saha, S. RoboAnalyzer: Robot visualization software for robot technicians. In *Proceedings of the Advances in Robotics*, AIR '17, pages 26:1–26:5, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5294-9. doi: 10.1145/3132446.3134890. URL `http://doi.acm.org/10.1145/3132446.3134890`.

[7] S. Hernandez-Mendez, C. Maldonado-Mendez, A. Marin-Hernandez, H. V. Rios-Figueroa, H. Vazquez-Leal, and E. R. Palacios-Hernandez. Design and implementation of a robotic arm using ros and moveit! In *2017 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, pages 1–6, Nov 2017. doi: 10.1109/ROPEC.2017.8261666.

[8] A. Jáger, T. Čechura, and M. Švejda. Non-standard robots for NDT of pipe welds. In *2018 19th International Carpathian Control Conference (ICCC)*, pages 196–200, May 2018. doi: 10.1109/CarpathianCC.2018.8399627.

[9] W. Khalil and E. Dombre. *Modeling, Identification and Control of Robots*. Kogan Page Science paper edition. Elsevier Science, 2004. ISBN 9780080536613. URL `http://books.google.cz/books?id=nyrY0Pu5kl0C`.

[10] W. Khalil and J. Kleinfinger. A new geometric notation for open and closed-loop robots. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 1174–1179, Apr 1986. doi: 10.1109/ROBOT.1986.1087552.

[11] MoveIt. *MoveIt Motion Planning Framework*, 2019. URL `https://moveit.ros.org/`.

[12] Omron. Programmable multi axis controller, CK3E Series, NY51-A Series, 2019.

[13] L. Sciavicco and B. Siciliano. *Modelling and Control of Robot Manipulators*. Advanced Textbooks in Control and Signal Processing. Springer London, 2000. ISBN 9781852332211. URL `http://books.google.fr/books?id=v9PLbcYd9aUC`.

[14] B. Siciliano. Kinematic control of redundant robot manipulators: A tutorial. *Journal of Intelligent and Robotic Systems*, 3(3):201–212, 1990. ISSN 0921-0296. doi: 10.1007/BF00126069. URL `http://dx.doi.org/10.1007/BF00126069`.

[15] S. S. Sinha, R. G. Chittawadigi, and S. K. Saha. In *The 5th Joint International Conference on Multibody System Dynamics (IMSD2018), Lisbon, Portugal*.

[16] RoboDK Software S.L. *RoboDK simulator for industrial robots*, 2019. URL `https://robodk.com/`.

[17] REX Controls s.r.o. *REXYGEN (Programming Automation Devices without Hand Coding)*, 2019. URL `www.rexygen.com`.

[18] Energid Technologies. *Actin, Software for Robotics Simulation and Control*, 2019. URL `www.energid.com/actin`.

[19] M. Švejda. *Optimization of robot architectures (in Czech)*. PhD thesis, University of West Bohemia, 2016. URL `http://home.zcu.cz/~msvejda/_publications/2016/4_SvejdaMartin_thesis_2016_06_14.pdf`.

[20] M. Švejda. Virtual simulation models for ADRA-2I NDT robot. Technical report, University of West Bohemia, 2018. URL `http://home.zcu.cz/~msvejda/_publications/2018/02_ROBIN_virtSimModel.pdf`.

[21] D. Youakim, P. Ridao, N. Palomeras, F. Spadafora, D. Ribas, and M. Muzzupappa. MoveIt!: autonomous underwater free-floating manipulation. *IEEE Robotics Automation Magazine*, 24(3):41–51, Sep. 2017. ISSN 1070-9932. doi: 10.1109/MRA.2016.2636369.