Automatic Configuration of the Dynamic Model for Common Industrial Robots

Michael Wenz, Heinz Wörn

Institute for Process Control and Robotics (IPR)
Universität Karlsruhe (TH)
D-76128 Karlsruhe, Germany
{mwenz,woern}@ira.uka.de

Abstract: Robot control software inherently depends on the mechanical structure of a given robot making it difficult to reuse and to share software among different robot types. Therefore motion control systems come in very large diversities. A specific goal of the project DodOrg (Digital On-demand Computing Organism for Real-time Systms) within the priority programme 1183 "Organic Computing" is the development of an organic robot control system. This is a generalized and self-configuring control system applicable to the motion control of a wide class of different robot types. Especially it automates the solving of the clumsy kinematic and dynamic equations necessary for motion control and offers functionalities for automate trajectory generation. This paper focuses on the dynamic robot model and describes how to automatically derive the dynamic equations from a declarative specification of the robot as well as the graphical user interface by which this process is invoked. In particular we generate the complete Lagrangian dynamic model for general serial manipulators.

1 Introduction

Industrial manipulators are extremely complex dynamic systems used to perform repetitive tasks such as assembly, arc and spot welding, machine loading and unloading, material handling, clueing, palletizing, polishing, cutting and many more. However not each robot is right for a given task, e.g. a Scara is not suitable for spray painting. Thus there exists a large variety of different kinematic structures and geometries. Some manipulators are designed for carrying heavy loads and others are optimized for very fast end effector movement. Indeed, the geometry of a robot and its number of degrees of freedom play a central role in all aspects of motion control software.

The different motion control systems of many vendors are often based on proprietary solutions. Then, developing and customizing software for specific kinematics require great efforts in terms of domain knowledge, time and cost. For each serial-link robot manipulator the kinematic and dynamic models have to be determined. These models are difficult to derive and computationally expensive. The kinematic model reflects the geometric and time based properties of motion without regard of the forces or moments

that cause the motion [GBF85]. Especially it describes the relationship between joint space and Cartesian space. The determination of the kinematic equations is done in two phases. At first the direct kinematics problem is solved and then the inverse kinematics problem. The direct kinematics model computes the resulting position and orientation of the tool centre point (TCP) when the robot's joint variables are given. This problem has a unique solution and is in general easily solved. Of more importance in motion control is the inverse kinematics model which computes the joint variables given a desired position and orientation of the robot's TCP. This problem is much more complicated, because a highly coupled nonlinear equation system has to be solved.

Elaborate control systems also have to take into account the manipulator dynamics. Besides the model-based control other applications of a dynamic model are in robot design and in simulation [SV89]. The dynamic model describes the relationship between the motion of the manipulator and the internal and external forces and torques that arise during its operation, e.g. inertial, centrifugal, Coriolis, gravitational and actuating torques or forces. The dynamic model consists of a set of nonlinear coupled second-order differential equations. The conventional way to develop the motion control software of a given robot is to do it with an engineering tool such as MatLAB™. With the dissemination of such tools, it has been easier to produce and to solve the kinematic and dynamic expressions than dealing with them manually. But they are still produced for a particular robot and have to be recomputed for other robot types. Furthermore these solutions are difficult to integrate in an existing environment and their portability is often restricted to C / C++ or Fortran code.

In the next section we review the state of the art and its relationship to our research. Section 3 introduces our project about the development of an organic robot control which includes so-called organic computing characteristics like self-configuration, self-healing and self-optimization features. The algorithms used to construct the dynamic robot model are then illustrated and the correspondent software implementation is detailed. We conclude in section 4 with a summary and a description of future steps.

2 State of the Art

Robot control software tends to be a big monolithic and proprietary software system that is difficult to enhance or to adapt to different robot types. In most cases it is vendor specific. With exception of some communication buses no standardisation efforts could prevail at all in the robot control domain. On the other side more openness of control systems is necessary in order to be able to rapidly integrate new functionalities into these systems, especially when the application requirements change. As a consequence a series of attempts were started to specify a universal architecture for developing robotic control software with open interfaces. Moreover software reuse and maintainability is generally improved by standardized interfaces imposed by these frameworks.

OROCOS (Open Robot Control Software / Open Real-time Control Services) is an object-oriented and robot independent framework for all aspects of robot control [Br02]. Therefore it comes with a general purpose library of open source software modules.

OROCOS is free software licensed under the GNU LGPL and written in C++. The control services run under RTAI with hard real-time properties, but also in "user-space" without guaranteeing real-time properties. However, OROCOS is also bound to these operating systems. It does not consider the dynamics of a manipulator and there are currently no dynamics algorithms implemented.

The Matlab Robotics Toolbox [Co95] is a freely available toolbox mainly used for education and teaching purposes. It uses a numerical technique to solve the inverse kinematics problem. The drawback of those iterative approaches is that they are slow for real-time applications and that they are unable to find all solutions. For the computation of the dynamical robot model the toolbox implements the recursive Newton-Euler algorithm [LWP80]. But the toolbox cannot provide any symbolic solutions.

Cosimir (Cell oriented simulation of industrial robots) is a robot and workcell simulation system developed by the Institute of Robotics Research at the University of Dortmund [FPW01] and currently distributed by Festo Corporation. This commercial version is very expensive and is primarily designed for industrial simulation of work flows with robotic systems and includes a library of robots from many different vendors. Cosimir does not compute the dynamic robot model. In all of the mentioned state of the art approaches there are no self-configuration or other organic computing features realized.

3 Organic Robot Control

In order to accommodate to different robot geometries it is necessary to automate the building of motion control software eliminating manual processes. DodOrg (Digital On-Demand Computing Organism for Real-Time Systems) granted by the German Research Foundation within the priority programme "Organic Computing" is a project to automatically configure robot control software. The main project goal is to fulfil the demands of self-configuration in the complicated robotics domain requiring minimal human involvement. Therefore robot control software is broken down into different components allowing code reuse and integration. The robot control system is the brain of the robot. By using self-configuration methods the time and costs for developing new robot control systems will significantly be reduced. Our approach is to perform symbolic calculations to automatically derive mathematical expressions used for kinematic and dynamic computations. Since it is possible to work with only symbolic data the equations are generated in full generality. These equations can on the fly be converted to code for the use in real-time control. This process is initiated by a newly developed configurator which runs directly on the robot control and opens up a large number of different selection and combination possibilities, each of which specifies a different robot type. In particular it is possible to describe how a manipulator is composed of a series of concatenated joints and links in order to create a kinematic chain. Among other things the user can specify:

- Type of joints (joints can either be revolute or prismatic), their number (a robot must contain at least four joints), their arrangement and constraints concerning their movement (e.g. maximum velocity and acceleration)

- Kinematic parameters like link lengths and link offsets are either directly provided by the manufacturer or can be measured on the robot and adjusted via calibration

- Dynamics data for each link: mass (one value: m_i), location of the centre of mass (three Cartesian coordinates related to the i^{th} frame: r_{xi}, r_{yi}, r_{zi}) and inertia tensors (six values: I_{xx}, I_{yy}, I_{zz}, I_{xy}, I_{xz}, I_{yz}). The first three values are the mass moments of inertia and the last three values are the mass products of inertia. To identify the dynamic parameters for a concrete robot the parameters are measured or estimated and a series of test motions is normally done.

Some of the selected combinations are not allowed or not desirable. Furthermore, some selected features impose constraints or requirements on other features, e.g. the kinematic structure of a SCARA requires that the three principal joints are parallel and that two of them are rotatory and the third one is a translational joint (abbreviated as RRT, RTR or TRR). Therefore we complement the configurator with a knowledge base about valid and invalid combinations. The configurator also comes with a library of reusable components having the same specified interfaces. So components that are common to robot control systems don't have to be reimplemented.

The automatic derivation of the kinematic model for a general robot is illustrated in [WW06]. Our approach is to automatically compute the Denavit-Hartenberg (DH) parameters from the declarative specification of the robot. The DH parameters describe the geometric relationship between the various joint-link-pairs of the manipulator. By determining the transformation matrices for each joint and subsequently for the whole robot, the symbolic expressions of the forward kinematics equations are automatically generated. The inverse kinematics is obtained by analytic reasoning from the direct kinematics equations in case that the analytical model of the robot exists. In the rest of the paper we concentrate on algorithms for calculating a closed-form solution of the dynamic equations of a given robot, which do not require iteration.

3.1 Lagrangian equations

The equations that describe the manipulator arm dynamics are discussed in this subsection. There exist several techniques for deriving the dynamics equations. As final result, all of them provide a set of equivalent equations. While Newton's equations treat each rigid link separately and also calculates unwanted interaction forces, Lagrange treats the system as a whole, typically yielding in simpler equations. The Newton-Euler formulation should be preferred for real-time computation for both simulation and control. However it is a recursive formulation. We are much more interested in a closed form solution in order to get a better insight into the structure of the equations for a given robot. Our approach is to follow the Lagrangian formulation, which is based on the differentiation of energy terms with respect to systems variables and time. The resulting equations can be calculated in closed form and allow an appropriate design of mechanical structures. The Lagrangian (L) equation is defined as the difference of the kinetic (K) and potential (P) energies. We assume that the potential energy is generated by gravity effects.

$$\tau_i = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \frac{d}{dt}\left(\frac{\partial K}{\partial \dot{q}_i}\right) - \frac{\partial K}{\partial q_i} + \frac{\partial P}{\partial q_i} \quad (i = 1,...,n) \qquad (1)$$

These equations relate the joint torque τ_i at the actuators as a function of joint positions, velocities and accelerations. We also use this symbol for the linear joint forces of prismatic joints. The mass is concentrated in a point. For a concrete robot the mass and the mass distribution of each link are given. The inertia tensor describes how the mass of the link is distributed with respect to a specified frame (here A). In general the inertia tensor is a symmetric 3×3 matrix of the form:

$${}^{A}I = \begin{pmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{pmatrix} \qquad (2)$$

In practice the inertia tensor is usually computed about the centre of the mass. With this choice, we define a generalized mass matrix for the i^{th}-link with m_i as the mass of link i and I as the 3×3 identity matrix. This matrix is time-invariant.

$$M_i = \begin{pmatrix} m_i * I_{3\times 3} & 0 \\ 0 & {}^{A}I_i \end{pmatrix}_{6\times 6} \qquad (3)$$

The kinetic energy consists of the kinetic energy of translation and the kinetic energy of rotation. We obtain the kinetic energy of a robotic manipulator by following equation:

$$K = \frac{1}{2}\dot{Q}^T \left(\underbrace{\sum_{i=1}^{n} \left(J_i (T_{0,i} * \begin{pmatrix} I_{3\times 3} & \vec{r}_i \\ 0 & 1 \end{pmatrix}) \right)^T * M_i * J_i (T_{0,i} * \begin{pmatrix} I_{3\times 3} & \vec{r}_i \\ 0 & 1 \end{pmatrix})}_{M(Q)} \right) \dot{Q} \qquad (4)$$

The kinetic energy of a system is a quadratic term and therefore positive unless the system is at rest. J_i is the i^{th} partial Jacobian matrix (dimension is 6×n) mapping the joint variables $q_1,\ldots,q_i$ to the inertial velocity of the centre of mass of the i^{th} link. The transformation matrix $T_{0,i}$ describes the i^{th}-frame relative to the robot's base frame. The vector $r_i=(r_{xi}, r_{yi}, r_{zi})$ gives the position of the centre of the mass of the i^{th}-link with respect to the i^{th}-frame. M(Q) is called the n×n manipulator mass matrix of the entire serial chain. It is symmetric and positive definite, and is therefore always invertible. We obtain the complete potential energy as the sum of the potential energy of each link, which are a function of the mass and the geometry of the manipulator.

$$P = \sum_{i=1}^{n} - m_i * (g_x, g_y, g_z, 0) * T_{0,i} * (\vec{r_i}, 1)^T \qquad (5)$$

The gravity vector (g_x, g_y, g_z, 0) is expressed in the robot's base frame. For most robots with the base mounted vertically on the floor the gravity vector is (0, 0, -g, 0) with g=9.8062 m/s^2 representing the gravitational acceleration. Putting the equations together we obtain the complete dynamic equations of a manipulator:

$$\begin{pmatrix} \tau_1 \\ \vdots \\ \tau_n \end{pmatrix} = M(Q) * \underbrace{\ddot{Q}}_{\frac{\partial \dot{Q}}{\partial t}} + \underbrace{\underbrace{\left(\sum_{i=1}^{n} \frac{\partial M(Q)}{\partial q_i} \dot{q}_i \right)}_{\frac{\partial M(Q)}{\partial t}} * \dot{Q} - \frac{1}{2} \begin{pmatrix} \dot{Q}^T \frac{\partial M(Q)}{\partial q_1} \dot{Q} \\ \vdots \\ \dot{Q}^T \frac{\partial M(Q)}{\partial q_n} \dot{Q} \end{pmatrix}}_{C(Q,\dot{Q}) * \dot{Q}} + \underbrace{\begin{pmatrix} \frac{\partial P}{\partial q_1} \\ \vdots \\ \frac{\partial P}{\partial q_n} \end{pmatrix}}_{G(Q)} \qquad (6)$$

C(Q, $\dot{Q}$) is a n×n matrix of centrifugal and Coriolis terms and G(Q) is a n×1 vector of gravity terms. M(q) and G(q) depend only on joint positions, while the Coriolis matrix depends on both joint positions and joint velocities.

3.2 Programming Interface

All dynamics components implement the same interface. So it is easy to integrate new dynamics components or to substitute a component with another one. The following code fragments describe the interfaces of the dynamics components that are automatically produced from the equations of the previous section. The usage is very simple, because the direct and inverse dynamics is performed by calling these functions. The initialization of the dynamics parameters of a robot arm, e.g. masses of the links, is done in another method.

The direct dynamics computes the movement of the manipulator due to external and internal forces/torques. The inputs to the direct dynamics computation are the current joint positions and velocities and the joint force/torque values. The outputs are the resulting joint accelerations. The method returns true if the accelerations could be calculated for each joint of the manipulator and otherwise false.

```
virtual bool directDynamics
   (const Vector &positions, const Vector &velocities,
    Vector &accelarations, const Vector &torques) const=0
```

Simulation requires solving the dynamic equations for acceleration. This method is useful for simulation the motion of a robot arm. The inverse dynamics computes the forces and torques which are necessary to perform a certain motion. The inputs to the inverse dynamics computation are the joint positions, velocities and accelerations over a specified trajectory. The outputs are the corresponding joint force/torque values that

have to be imposed to the robot to follow the trajectory. This method is useful for determining the set-points of the actuators that are required for a desired motion and is a starting point for model-based control.

```
virtual bool inverseDynamics
   (const Vector &positions, const Vector &velocities,
    const Vector &accelarations, Vector &torques) const=0
```

3.3 Robotics Example

The automatic derivation of the dynamic equations has been successfully implemented. As an illustration we exemplarily now consider a Scara robot arm. The robot has four degrees of freedom in the Cartesian space. The link parameters required to calculate the dynamic equations are mass m_i, location of the centre of gravity and the terms of the inertia tensor as well as the Denavit-Hartenberg parameters describing the robot geometry. The length of arms 1 and 2 are abbreviated as a_1 and a_2; the variables of the first and second joint as θ_1 and θ_2 and the centre of mass of the first and second link as r_{x1} and r_{x2}. The procedure used to derive the dynamic model entails the steps: symbolic generation of the kinetic energy equations, symbolic generation of the potential energy equations and computation and differentiation of the Lagrange functions. For this specific robot the potential energy is a constant since it does not depend on the joint variables. We write the dynamic equations more compactly in its standard form. Because of the limitation of space, the details of the derivation are not included here. The complete robotics manipulator dynamic model is given by:

$$\begin{pmatrix}\tau_1\\ \tau_2\\ \tau_3\\ \tau_4\end{pmatrix} = \underbrace{\begin{pmatrix} m_{11} & m_{12} & 0 & m_{14}\\ m_{21} & m_{22} & 0 & m_{24}\\ 0 & 0 & m_{33} & 0\\ m_{41} & m_{42} & 0 & m_{44}\end{pmatrix}}_{M(Q)} * \ddot{Q} + \underbrace{\begin{pmatrix} c_1\\ c_2\\ 0\\ 0\end{pmatrix}}_{C(Q,\dot{Q})*\dot{Q}} + \underbrace{\begin{pmatrix} 0\\ 0\\ -(m_3+m_4)*g\\ 0\end{pmatrix}}_{G(Q)}$$

$$m_{11} = \begin{array}{l} m_1 r_{x1}^{\,2} + m_2(r_{x2}^{\,2} + a_1^{\,2}) + 2a_1 r_{x2}\cos(\theta_2)m_2 + \\ (m_3+m_4)(a_1^{\,2} + 2a_1a_2\cos(\theta_2) + a_2^{\,2}) + \sum_{i=1}^{4} I_{xxi}\end{array}$$

$$m_{12} = m_2(r_{x2}^{\,2} + a_1 r_{x2}\cos(\theta_2)) + (m_3+m_4)(a_2^{\,2} + a_1 a_2\cos(\theta_2)) + \sum_{i=2}^{4} I_{xxi} = m_{21}$$

$$m_{22} = m_2 r_{x2}^{\,2} + a_2^{\,2}(m_3+m_4) + \sum_{i=2}^{4} I_{xxi}$$

$$m_{33} = m_3 + m_4$$

$$m_{44} = I_{xx4} = -m_{14} = -m_{41} = -m_{24} = -m_{42}$$

$$c_1 = -(m_2 r_{x2} + a_2(m_3+m_4))(2\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_2^{\,2})a_1\sin(\theta_2)$$

$$c_2 = (m_2 r_{x2} + a_2(m_3+m_4))\, a_1\sin(\theta_2)\,\dot{\theta}_1^{\,2}$$

4 Conclusion

It is not practical or advisable to derive the dynamic model of each robot manipulator manually and case by case. Therefore in this paper, a methodology for automatically deriving the Lagrangian equations has been illustrated. In particular we highlight the self-configuration and symbolic processing features of our system as well as the automated generation of the dynamics equations given a description of the controlled robot. Due to the analytical nature we get a more general result and can automate this process while numerical methods are often specific to a certain problem. Moreover we also automatically document how the solutions were found.

The most commonly used equations to model the dynamics of a robot are the Lagrange and Newton-Euler formulations. Here we used the Lagrangian equations, because they are straightforward to derive. Closed-form solutions can also be better parallelized than recursive ones and are especially suited for multi-core technology. Our major goal includes the specification of a modular software architecture for motion control which can be easily configured. Work is ongoing to improve the system and to integrate further functionalities, e.g. we will also implement generalized axis controllers that keep the actual actuators values as close as possible to the corresponding set points by means of linear and nonlinear control algorithms.

Acknowledgements. The work on which this article is based is granted by the German Research Foundation (DFG) within the scope of the priority programme 1183 (see http://www.organic-computing.de/spp/).

References

[Br02] Bruyninckx, H.: A Free Software Framework for Advanced Robot Control, Proceedings of the 7th ESA Workshop on Advanced Space Technologies for Robotics and Automation (ASTRA 2002), Noordwijk, The Netherlands, November 19 - 21, 2002.

[Co95] Corke, P.I.: A computer tool for simulation and analysis: the Robotics Toolbox for MATLAB, Proceedings of the 1995 National Conference of the Australian Robot Association, Melbourne, Australia, pp. 319-330, July 1995.

[FPW01] Freund, E.; Pensky, D.; Wischnewski, R.: Cosimir: Open 3D Simulation System for Production Automation Processes, in Proceedings of the 10th International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD'01), pp. 16-18, Vienna, Austria, May 2001.

[GBF85] Goldenberg, A.A.; Benhabib, B.; Fenton, R.G.: A Complete Generalized Solution to the Inverse Kinematics of Robots, IEEE Journal of Robotics & Automation 1(1), pp. 14-20, March 1985.

[LWP80] Luh, J.Y.S.; Walker, M.W.; Paul, R.P.C.: Online computational scheme for mechanical manipulators, ASME Journal of Dynamic Systems Measurement and Control, 102:69–76, 1980.

[SV89] Spong, M.; Vidyasagar, M.: Robot Dynamics and Control, New York, John Wiley & Sons, 1989.

[WW06] Wenz, M.; Wörn, H.: Rule based Generation of Motion Control Software for General Serial-Link Robot Manipulators, Proceedings of the 8th International Workshop on Computer Science and Information Technologies (CSIT'2006), Germany, 2006.