Gravity Compensation Control of da Vinci Robot

Alex Rutfield, Amaid Zia, Brandon Boos, Junius Santoso, Nuttaworn Sujumnong
Department of Robotics Engineering
Worcester Polytechnic Institute, Worcester, MA, USA

***Abstract*—This paper presents work done on the da Vinci Surgical System. The authors derived symbolic forward kinematics and stationary dynamics equations for the master tool manipulator with the objective of implementing accurate gravity compensation. Unknown parameters necessary for evaluating the symbolic equations were determined using a least squares approach on recorded joint position and torque data. The symbolic equations were also implemented in simulation. Additionally, the authors built the patient side manipulator Gazebo model. Results of both simulation and physical implementation of gravity compensation are shown. The physical gravity compensation accurately maintained the position of the manipulator but had difficulty maintaining the correct orientation.**

I. BACKGROUND

A. Introduction

Robots are becoming increasingly involved in surgeries, and many surgeons have found them helpful in a variety of operations. However, these robots must be able to move exactly as the surgeon desires, using high accuracy and precision, to be successful and usable in surgeries. As a result, each surgical robot must not move unless the operator requires it to, and the operator should not have to exert more torque into the system than they need to. This is the case for the da Vinci Surgical System, a robot created by Intuitive Surgical for minimally invasive surgery. The da Vinci robot consists of two arms: the master tool manipulator (MTM), or master arm, which the operator controls, and the patient side manipulator (PSM), or slave arm, which operates on the patient directly. The movements of the MTM are used to control the PSM.

The Worcester Polytechnic Institute AIM lab has been working on modeling the da Vinci in the Robot Operating System (ROS) and attempting to improve the user's control. One way to do so is to supply the arms with accurate gravity compensation, allowing them to remain in the last position to which the operator moved them. In the case of the master arm, gravity compensation will also allow the operator to control the robot without having to supply any force to compensate for gravity themselves. This feature would allow the operator to control the system much more easily and accurately during a procedure.

Currently, the lab has a model of the MTM and is using PID to control it. It also does not have a model of the PSM. The purpose of this project was to improve the model and control of the robot by finding the gravity compensation, adding it to the master arm model, and then creating the model for the PSM. The team found the gravity compensation by finding some of the arms unknown parameters and using the Euler-Lagrange approach to find the desired torque. We then tested the parameters on the actual robot and attempted to use them to control the MTM in Gazebo simulator. Finally, we created the PSM model so similar projects could be done to improve the control in the future.

II. LITERATURE REVIEW

A. Da Vinci Surgical Robot

The da Vinci Surgical Robot is an advanced surgical robot system that allow surgeons to perform operations on patients with extreme precision and stability. With a master-slave control system, the surgeon can perform an operation from a distance through the surgeon console. The master system is equipped with 3D HD vision system and 8 degrees-of-freedom intuitive manipulators with built-in communication facility. The surgeon console provides full control of the EndoWrist, a highly precise 7 degrees-of-freedom manipulator on the patient side that is equipped with a 3D HD camera and provides natural motion, dexterity of a human hand, and minimization of the operator's hand tremor.

B. Previous Work

With collaboration of researchers from various institutes such as John Hopkins University, Worcester Polytechnic Institute, and Intuitive Surgical. Inc.; an open-source research tool kit for the da Vinci robot has been developed. This tool kit is based upon the packages from Robot Operating System (ROS) and the Surgical Assistant Workstation (SAW). ROS includes libraries for teleoperation, hardware interface, and system control between master system and slave system, while SAW includes libraries for both real-time robot control and real-time computer vision [1].

Previously, the AIM Lab, sponsored by WPI, had been working on the tools manipulation and path recognition implementation for master-slave system of the da Vinci robot. This development allows the researchers to study and analyze the robot's workspace and dexterity while it is moving and avoiding obstacles during the experiment. In addition to the algorithm, the team also created CAD models for the system and incorporated RRT* planners, 3D point cloud and streaming for feedback. This was done in order to simulate and study the robot's trajectory through ROS [2].

C. Parameter Identification

During the implementation of gravity compensation, parameter identification is a crucial step that must be resolved at the very beginning of the project. In order to identify the parameters of the da Vinci robot, we have studied several documents for methods and best solutions that are suitable with the system. According to Wu et al. [3], there are two methods of parameter identification: off-line method, which is involved with pre-analysis data collection, and on-line method, which requires real-time data update while the robot is operating. Using any of the suggested methods, the identification procedure is still mainly based on the Lagrangian and Euler-Lagrange equations. These equations are rearranged into linear equations in order to implement the least squares estimation method to solve for the dynamic parameters. A further explanation on one method of identifying dynamic parameters for robot is suggested by Bao et al. [4]. This method is to rearrange all unknown parameters into vector form (represented with ϕ). The equation is represented below:

$$\phi = (W^T W)^{-1} W^T \tau \tag{1}$$

where W represents an n x m kinematic matrix of the arms' motion and τ represents an n x 1 vector of forces and torques at each joint.

III. METHODOLOGY

A. *Gravity Compensation*

The equation governing the dynamics of a robotic system is given by:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau \tag{2}$$

Where $M(q)$ is the inertia matrix, $C(q, \dot{q})$ is the vector of centrifugal and Coriolis forces, and $G(q)$ is the gravitational forces vector [4].

B. *Forward Kinematics*

In order to determine the G matrix, an accurate mathematical model for the system must be created. The da Vinci system MTM consists of seven revolute joints and one pinching joint. For the purpose of the gravity compensation, the pinching joint is ignored. The first step towards creating this model is to perform the forward kinematics analysis of the system.

After identifying the links and joints in a simplified diagram of the arm at the home position, the team assigned frames using Denavit-Hartenberg parameters (Fig. 1 and TABLE I)

TABLE I
DH PARAMETERS FOR MTM

Joint #	θ	d	α	a
1	q_1	$-L_1$	$-\frac{\pi}{2}$	0
2	$-q_2 + \frac{\pi}{2}$	0	0	L_2
3	$-q_3 - \frac{\pi}{2}$	0	$\frac{\pi}{2}$	L_3
4	q_4	L_4	$-\frac{\pi}{2}$	0
5	$-q_5 + \pi$	0	$-\frac{\pi}{2}$	0
6	$q_6 - \frac{\pi}{2}$	0	$\frac{\pi}{2}$	0
7	$-q_7$	0	0	0

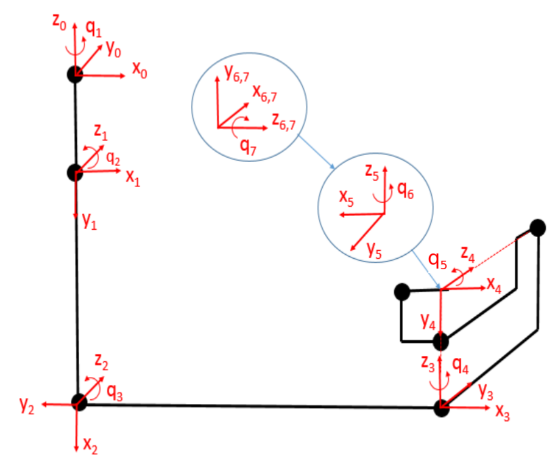

Fig. 1. Frames assignment of master arm using DH parameters.

The home position is the position at which all joint positions are zero. Because the existing software assigned the positive joint directions, the team ensured that the DH parameters accurately reflected that assignment. Additionally, the team used symbolic representation for all lengths. The length parameters were difficult to accurately measure and were thus included in the parameter identification task. Utilizing previous work, the team used a MATLAB function to obtain the 4 x 4 homogeneous transformation matrices for each frame with respect to the base frame from the DH parameters.

C. *Lagrangian*

The next step in determining the G matrix is to compute the Lagrangian of the system. The Lagrangian will be utilized in the Euler-Lagrange approach to forward dynamics of robot manipulators. The Lagrangian of a system is given as:

$$L = K - P \tag{3}$$

where K is the kinetic energy of the system and P is the potential energy. Because gravity compensation only account for torques in a stationary system, the kinetic energy term can be eliminated.

$$L = -P \tag{4}$$

The potential due to gravity is given as

$$P = g \sum_{i=1}^{n} m_i h_i \tag{5}$$

where g is the acceleration due to gravity, n is the number of masses, m is the mass, and h is the distance along the axis of g between the mass and the origin of the base frame. The masses were left as symbolic parameters to be identified. The h terms were determined using the transformation matrices from the forward kinematics. The h terms are functions of the robots pose.

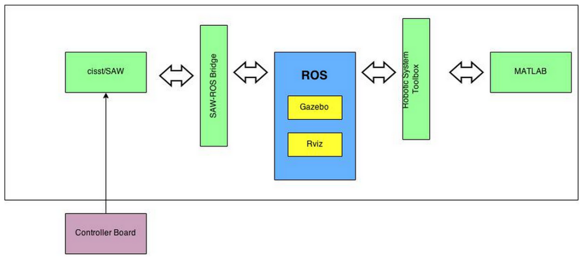

Fig. 2. Hardware & Software Architecture of da Vinci Robot at WPI.

D. *Euler-Lagrange Equation*

The Euler-Lagrange equation allows for the calculations of generalized forces (torques or forces) from the derivatives of the Lagrangian.

$$\tau = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} \tag{6}$$

Once again, this can be simplified by identifying terms that go to zero for a stationary manipulator.

$$\tau = -\frac{\partial L}{\partial q} \tag{7}$$

Using equation 4 and 7, the symbolic torque terms necessary for gravity compensation can be determined.

E. *Setup*

The setup consisted of da Vinci MTM connected to the controller boards which reads the data from the MTM and communicate it to a ROS terminal. This ROS terminal has previously been setup with most of the software framework for the da Vinci robot. This terminal was further connected through a wireless link to a terminal running MATLAB with Robotic System Toolbox (RST). RST allows the team to read and write to the ROS topics.

F. *Data Collection Issues*

After developing the mathematical model the next step was to collect the corresponding torques and joint position data which will help us in estimating the parameters necessary for gravity compensation. As the team started with the task, several issues arose:

1) *Communication issues*
The first issue that the team faced was that whenever a wireless device (like a laptop) was involved in the communication between ROS and the MATLAB RST terminal there were communication issues that prevented the device recognition for both the terminals. The quick fix for this was to modify the operating system's hosts file (for Linux it is located under home/etc/hosts) by adding the ip address and device name of the computer to which the system is attempting to communicate.
2) *Unstandardized message (msg) types*
The next issue that the team confront was that the already establish ROS framework for the da Vinci utilized some non-standardized msg types which are not readable in MATLAB. e.g. The Torque data had the msg type cisst/vecDoublev which MATLAB was not able to read. To solve this issue, the team implemented a ROS node that was able to create a new standardized topic type, read data from the unstandardized topic type and copy it to the new topic. This new topic was then read in MATLAB.
3) *Workspace restriction*
Another issue was that the operating space for the MTM was restricted by the frame of the robot in the laboratory. Consequently, the team was unable to operate the arm over the full range of its joints. If attempted, the arm would have crashed into the frame and potentially damaged the robot. This forced the team to work within the restricted workspace.
4) *Non-definitive joint limits*
For some of the joints, the joint limits are dependent upon the state of the previous joints. e.g. the limit of joint 3 was dependent upon the state of joint 2 . This caused the team to not have numerically constant hard limits for some of the joints. The team was able avoid the joint limits by the chosen method of collecting data.

G. *Data Collection Strategy*

Our data collection strategy was designed so that we could get position and corresponding torque data at varying range of arm configurations and avoid all joint limits. The strategy was as follows:

1) Place the MTM in a random pose by hand.
2) Record the joint positions for each pose in MATLAB.
3) Repeat steps 1 and 2 for desired number of poses
4) Command the arm (from MATLAB) to move to recorded pose
5) Record the torque values necessary to hold the pose
6) Repeat steps 4 and 5 for all recorded poses

By following the above method, the team was able to get many data sets for torque and joint position values.

H. *Regressor and Parameter Matrices*

The next step was to divide our torque equations into a matrix of knowns (called the Regressor Matrix Y) and the matrix of the unknowns (called the Parameter matrix π). The parameter matrix consisted of products of unknown link lengths, link masses and locations of center of masses. After splitting, the team got a 7x12 Regressor Matrix and 12x1 parameter matrix.

$$\tau = Y(q, \dot{q}, \ddot{q})\pi \tag{8}$$

I. *Least Square Method*

Now the team substituted each of our data sets into torque values and the regressor matrix. Then the team stacked these matrices on top of each other to form an equation as shown below

$$\bar{\tau} = \begin{bmatrix} \tau(t_1) \\ \vdots \\ \tau(t_N) \end{bmatrix} = \begin{bmatrix} Y(t_1) \\ \vdots \\ Y(t_N) \end{bmatrix} \pi = \bar{Y}\pi \tag{9}$$

From this equation, the team calculated the estimates of the parameters by doing the left pseudo inverse of our stacked regressor matrix

$$\pi = (\bar{Y}^T\bar{Y})^{-1}\bar{Y}\bar{\tau} \tag{10}$$

Now that the team had the estimates of parameters, the team could get an estimate of the torques required for gravity compensation at some pose by substituting the joint position values into the regressor matrix and multiplying it with the estimates of parameter matrix.

J. Modeling & Simulation

The gravity compensation control of da Vinci MTM arm was simulated using the Gazebo simulator by Open Source Robotics Foundation (OSRF) [5], as shown in Fig. 3. Gazebo has the capability to allow users to control each joint of the simulated robot through its internal controller. However, the team would be using the ros_control package instead since it will allow us to use our own developed controller. Implementation of ros_control package in Gazebo requires two main steps. The steps will be briefly described here, for more detail information refer to the ros control tutorial on Gazebo web page. The first step was to add the transmission elements into the Universal Robot Description Format (URDF) file of MTM. The transmission elements were responsible for actuating our robot joints. Joint name, type of transmission, and hardware interface need to be specified. The second step was to include gazebo_ros_control plugin into our URDF model as well [6].

Once the plugin has been included, the team then proceeded to create the gravity compensation controller package. Many details will be omitted here for the sake of conciseness, refer to the Gazebo tutorial page for additional details. A ROS python node was created for our gravity compensation control implementation. This node subscribed to the joint_state topic, calculated the required torque values for each joint based on the gravity compensation model and PID feedback, and finally published the torque commands to the joint(n)_controller/command topics. Furthermore, the desired joint configuration is also specified in this controller node, for this simulation we set zero radian for all joints as the desired configuration.

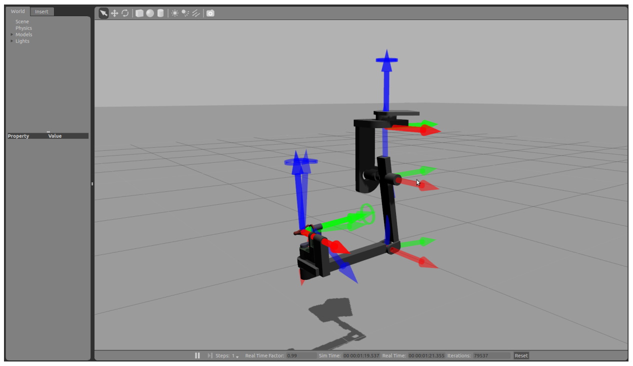

Fig. 3. MTM arm simulated in Gazebo with all active joints shown.

The PID feedback term was necessary since there was no a way, while simulating the torque control, to specify a position or hold the arm as done for the real robot. The control law is then given as:

$$\tau = G(q_m) + K_p(q_d - q_m) + K_v(\dot{q}_d - \dot{q}_m) + K_i \sum (q_d - q_m) \tag{11}$$

where q_d = q desired and q_m = q measured. Due to the way we defined the position and velocity error as $(q_d - q_m)$ and $(\dot{q}_d - \dot{q}_m)$ respectively, K_p, K_i, and K_v are positive definite. Furthermore, since we have seven joints to control, K_p, K_i, and K_d are 7x7 diagonal matrices. The $G(q_m)$ vector was derived from the forward kinematic and center of mass locations specified in the MTMs URDF file. The derivation would be similar to what was done for the real MTM. For the simulation, instead of estimating the parameters, the team used the parameters specified in the URDF file. For simplicity, all the masses are specified to be point masses and located at the joints. In the future, this parameter specifications can be updated with the estimated parameters found experimentally to improve the simulation model. Note that our work in this paper did not include all the estimated dynamics parameters. In the end, the team would obtain the gravity vector G(q) as a function of joint variables for our torque input.

The PID gains for the feedback term were tuned by first setting the K_v and K_i gains to be zero and adjusting the K_p gains until the system produced constant amplitude oscillation. We then slowly increased the K_v gains to damp the oscillation and finally increased the K_i to correct for any steady state offset. Without the feed forward gravity term, normally this PID gains would have large gains. In this case, since the PID loop is only used to correct for the model error, the gains were relatively small.

In this paper we will also present our work on building the Gazebo model of patient side manipulator (PSM) arm. This work involved creating the model using the Simulator Description Format (SDF) because URDF does not currently support joint loops (parallel linkages) which is the case for the PSM arm. The virtual and collision elements are specified using the mesh files of PSM arm found in jhu dvrk github repository. SDF format requires users to specify the poses of all the links and joints of the robot. The poses for each link and joint are obtained by assembling the PSM CAD model in SolidWorks first and then using the measure feature to obtain the origin of each link with respect to the reference frame which in this case is the origin of the psm_base_link.

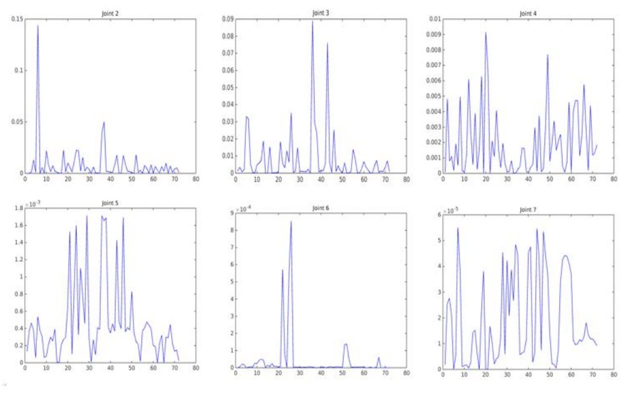
Fig. 4. Square of joint error.

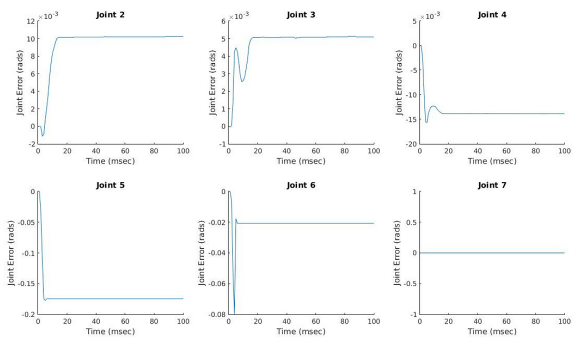
Fig. 5. Joint drift.

IV. RESULTS

A. Implementation on Real Robot

The team determined numerical values for all twelve symbolic parameters determined using the symbolic torque equations. Using these values, the team wrote a MATLAB code that applied gravity compensation for a given position. Only gravity compensation terms were included in the applied torque, without any error based controller. At a rate of roughly one millisecond, the code read the joint positions and applied an appropriate torque. The visual results can be seen in the video results of this project.

The errors between the torque values read at positions and the torque values calculated using the determined parameters is shown in Fig. 4 for joints 2 through 7. Joint 1 had a constant torque of zero when applying gravity compensation.

Additionally, the team measured errors in joint values after the manipulator was realized when gravity compensation was being applied Fig. 5. As can be seen, the joint errors were less than 15 milliradians for joints 2 through 4. Joint 5 and 6 show greater errors and Joint 7 shows roughly 0 error.

While the team was satisfied that the arm maintained position using only the gravity compensation for most configurations, the orientation rarely held accurately. That is, joints 2 through 4 maintained positions while joints 5 through 7 did so rarely or at least inconsistently. Because of the low mass and length values for the last few links, the parameters were

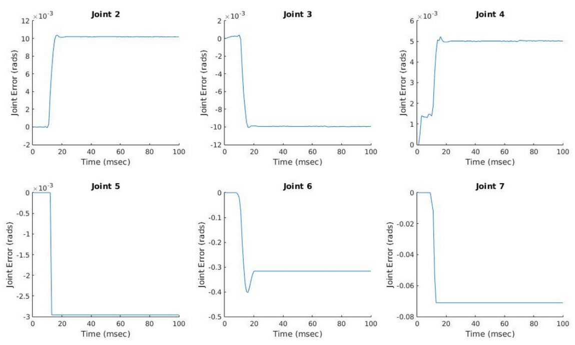
Fig. 6. Joint drift with some torques zeroed.

difficult to accurately determine. Regardless of the position when released, the joints 5 through 7 would move to a consistent position under gravity compensation. One solution to this issue is to set the torques for joints 5 through 7 to be zero during gravity compensation. This alleviates inaccurate torques which may be felt by an operator and allows for a small range of position to be held due to friction terms. The drift results for this condition can be seen in Fig. 6.

B. Modeling and Simulation

Fig. 7 below shows the progression of the seven joint values as a desired position input of zero radians for all joints was applied. It appears that an underdamped response was present and all the joint values eventually seemed to converge to the desired value. However, upon a closer look of the system response as shown in Fig. 7-bottom, the joint values did not exactly converge to zero, instead the values oscillated near

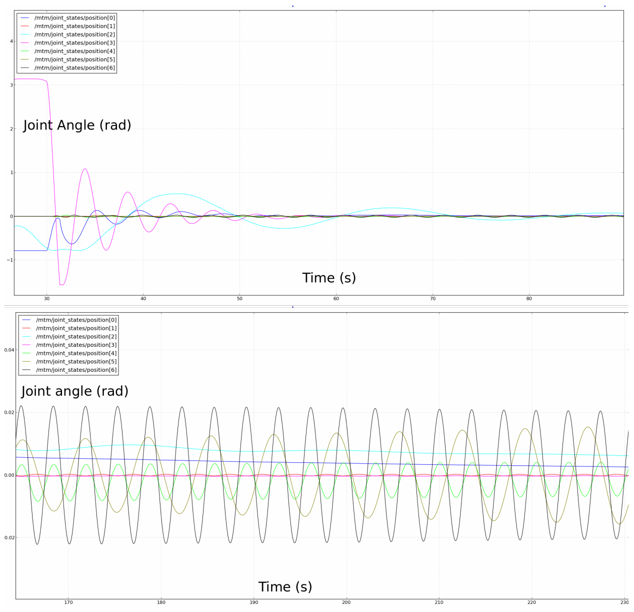

Fig. 7. Joint angles as progression of time, top - wider view, bottom - close up view.

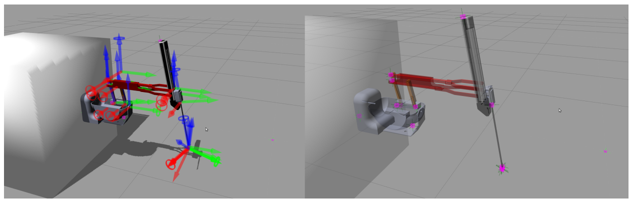

Fig. 8. left - PSM model with all joint locations shown, right - PSM model with center of mass locations.

zero. There is probably a need to better tune the gain values especially for the seventh joint, as the authors found it hard to keep this particular joint still. Nevertheless the team managed to obtain a stable system response.

The PSM model described in the previous section is shown in Fig. 8. The PSM model contains seven revolute joints and one prismatic joint. Currently, all the center of masses are located at the joints for the sake of simplicity. In the future, the model can be improved by adjusting the center of mass locations with a result of a parameters estimation. All the joints of the PSM model can be torque controlled as was done for the MTM arm. This can be beneficial for future work done related to the PSM arm.

V. Conclusion

The team was successful in implementing gravity compensation on the main links of the MTM. Using the parameters derived from the least square fit, the arm was able to support itself with the correct gravity compensation. Similar gravity compensation was applied to the Gazebo model of the MTM, enabling it to be controlled as well. Finally, the team created a model of the PSM in Gazebo that could be used on similar projects in the future. Overall, the project has provided a very strong basis into allowing the operator full and accurate control over the da Vinci arm.

References

[1] P. Kazanzides, Z. Chen, A. Deguet, G. S. Fischer, R. H. Taylor, S. P. DiMaio. *An Open-Source Research Kit for the da Vinci Surgical System*, 2014.

[2] Z. Z. G. S. F. Adnan Munawar, *Implementation of a Motion Planning Framework for the daVinci*, in Hamlyn Symposium on Medical Robotics, London, UK, 2014.

[3] W. Jun, W. Jinsong, Y. Zheng, *An overview of dynamic parameter identification of robots, Robotics and Computer-Integrated Manufacturing*, Volume 26, Issue 5, October 2010, Pages 414-419, ISSN 0736-5845, http://dx.doi.org/10.1016/j.rcim.2010.03.013.

[4] Ping-An Bao; Ping Jiang; Hui-Tang Chen, ”A learning scheme for the parameter identification of robot dynamics,” Industrial Technology, 1996. (ICIT ’96), Proceedings of The IEEE International Conference on , vol., no., pp.651,655, 2-6 Dec 1996

[5] M. Spong, S. Hutchinson and M. Vidyasagar, *Robot modeling and control*. Hoboken, NJ: John Wiley & Sons, 2006.

[6] K. Nate and H. Andrew. *Design and use paradigms for gazebo, an open-source multi-robot simulator*. In Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004), volume 3, pages 21492154, Sept 2004.

[7] Barros, Taiser Tadeu Teixeira; Fetter Lages, Walter, *A Mobile Manipulator Controller Implemented in the Robot Operating System*, ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of , vol., no., pp.1,8, 2-3 June 2014.