Dynaban, an Open-Source Alternative Firmware for Dynamixel Servo-Motors

Rémi Fabre, Quentin Rouxel(✉), Grégoire Passault, Steve N'Guyen, and Olivier Ly

Rhoban Football Club Team, LaBRI, University of Bordeaux, Bordeaux, France
{remi.fabre,quentin.rouxel}@labri.fr

Abstract. In this paper, we present an alternative open-source firmware for the Dynamixel MX-64 servo-motor. We discuss software features to fully exploit the hardware capabilities of the device. In order to enhance the default controller, a friction model and an electric model of the motor are embedded into the firmware. The parameters of the model are found using a black-box optimization algorithm. A feed-forward method is proposed to follow position, speed and torque trajectories. The approach is tested with a highly dynamic kick movement on our humanoid soccer robot *Sigmaban* whose torque trajectories are computed using a classic rigid body inverse dynamics. The comparison between the default control strategy and the proposed one shows significant improvements in terms of accuracy, delay and repeatability.

Keywords: Open-source · Model-based control · RoboCup · Humanoid robot

1 Introduction

Servo-motors offer a full turnkey solution for controlling electric motors and are heavily used in the field of robotics. Commercial servo-motors are the most common, but their proprietary firmware can't be enriched by the community and limits the realm of possible. The default control approach typically relies on angular feed-back and a Proportional-Integral-Derivative (PID) controller. This method is easy to use and offers good results when static positions are needed but is unsatisfactory when following complex trajectories which tend to be delayed and distorted. In particular, humanoid robots often call for precise, highly dynamic and torque-heavy movements. Achieving these with the out-of-the-box controller has proven to be difficult.

Instead of waiting for an angular error to appear, a feed-forward (FF) approach preemptively sends a command that produces the predicted torque to follow a given trajectory. To achieve that, the internal behavior of the motor must be modeled and the external torques and moments must be accounted for. The PID controller is only used to compensate unforeseen perturbations and model's inaccuracies.

S. Behnke et al. (Eds.): RoboCup 2016, LNAI 9776, pp. 169–177, 2017.
https://doi.org/10.1007/978-3-319-68792-6_14

In this paper we present an open-source firmware[1] for the Dynamixel MX-64 servo-motor that is fully compatible[2] with the default one. Advanced features and possible enhancements are discussed but the main focus is given to the embedded implementation of a feed-forward control strategy. A friction model and an electric model of the motor are chosen. A wide set of detailed measures is taken and the CMA-ES evolutionary algorithm is used to identify the model's parameters. As application, a mechanical model of the humanoid robot *Sigmaban* [1] is used and the inverse dynamics of a football kick are computed[3]. Finally, the kick is played on the robot and the impact of the method is quantified.

1.1 Related Works

Open projects related to servo-motors have been around for years. OpenServo[4] is a mature project that offers low cost electronic boards but uses an 8-bit micro-controller that lacks the computational power of the ARM cortex M3 found in the Dynamixel. The DDServo project[5] [7] provides open hardware and firmware for the RX-28 and RX-64 motors, advanced control techniques are discussed and simulated but are not implemented. The Morpheus firmware[6] is an alternative firmware for the AX12 servomotor. A detailed friction characterization was carried by [6] and was used to choose a friction model. A model of the AX12 servo-motor was detailed in [4]. Schwarz and Behnke [3] use an ideal DC motor model and friction model covering the Stribeck effect.

The presented adds the inertia of the motor and its gear-box which, given the high ratio of the gear-box, appears to be meaningful. The main novelty of the proposed approach is the integration of a model-based feed-forward control into the original micro-controller instead of using the servo-motor as a black-box. Moreover, an offline evolutionary algorithm[7] is used in order to fit the model to an arbitrary rich set of measures. The quality of the results is of an order of magnitude better than what was achieved by hand-tuning the parameters.

2 An Open Source Firmware

Current Sensing: An interesting announced feature of the MX-64 series is the current sensing. Since the current is proportional to the torque, one would think that the servo-motor can be torque controlled. Unfortunately, testing the servo-motor with its default firmware proved otherwise. A high precision resistor of $5\,\text{m}\Omega$ called *rSense* is connected in series with the motor. The resistor's voltage is amplified and read through an Analog to Digital Conversion (ADC) pin.

[1] Dynaban: https://github.com/RhobanProject/Dynaban.
[2] http://support.robotis.com/en/product/dynamixel/mx_series/mx-64.htm.
[3] Rigid Body Dynamics Library: http://rbdl.bitbucket.org/.
[4] OpenServo project: http://www.openservo.com/.
[5] DDServo project: https://github.com/wojtusch/DDServo.
[6] Morpheus firmware: https://actuated.wordpress.com/ax12firmware/.
[7] CMA-ES Library: https://www.lri.fr/~hansen/html-pythoncma/.

The current appears to be very noisy and asymmetrical with the sign of the rotational speed. External measures showed that the current sensing issues are hardware related. Much better results were recorded when the *rSense* was held a few centimeters away from the motor and connected with copper cables. In the presented work, the current measure is considered unreliable as a feed-back information with the default electronics and thus not used.

High Precision and High Frequency Measures: When recording a physical value from a servo-motor with the default firmware, the delay between the actual measure and the moment the read is performed through the serial bus is poorly known. Moreover, the frequency of the measure is dependent on the communication bus. An open firmware allows for very specific, hardware time-stamped, high frequency measures. For instance, it is possible to read the current position at a frequency of 10 kHz, store the values in RAM for the duration of the experiment and send the values to the user through the serial port.

Model Based Estimation of the Torque Produced by the Motor: While the current measure proved to be unreliable, it is still possible to estimate the produced torque with a model-based approach. Knowing the voltage applied by the Pulse Width Modulation (PWM) signal and knowing the current speed, the model can estimate the electrical torque created by the motor and the friction torque. These values are updated at 1 kHz and are available to the user through the communication protocol.

The average speed is used where the instantaneous speed is needed, therefore a strong trade-off between precision and delay is present. Taylor expansion based algorithms, and Kalman filter based algorithms [2] are a few options that may enhance the speed estimation in a future work.

Feed-Forward Trajectory Control Mode: The embedded electrical and friction models can be used to follow position, speed and torque trajectories simultaneously. Knowing $\omega(t)$, $\omega(t + dt)$ and $\tau_o(t + dt)$ the needed command voltage can be calculated:

$$u(t + dt) = k_e \times \omega(t) + \frac{R}{k_e} \times (\tau_o(t) + \tau_a(t) + \tau_f(t)) \tag{1}$$

$$\tau_a(t) = \frac{\omega(t + dt) - \omega(t)}{dt} \times I_0 \tag{2}$$

where ω is the rotational speed, τ_o is the output goal torque set by the user, τ_a is the torque needed to accelerate the rotating parts of the motor which moment of inertia I_0 is a model parameter (empty shaft, gear box) and τ_f is the friction torque given by the friction model. R and k_e are constants of the DC motor.

The user is responsible for providing ahead of time a position and a torque trajectory to the firmware. The torque trajectory provided is the torque needed to accelerate the objects attached to the motor and also needed to oppose the torques applied on the motor's shaft (weight, inertia, contacts). The firmware is responsible for deducing the voltage command needed to accelerate the motor rotating parts (shaft, gear box).

The chosen trajectory representation is polynomial. The trajectories are communicated to the firmware as a 4^{th} order polynomial for the trajectory position and a 4^{th} order polynomial (5 float values) for the torque trajectory. The speed trajectory is internally deduced by derivation of the first polynomial. Trajectories are typically 150 ms long although the duration is user-configurable for each trajectory. A second set of polynomials can be buffered into the firmware to allow for continuous transitions between several trajectories.

In this mode, the speed is never deduced from the encoder, making the model's contribution to the voltage command a full open-loop feed-forward approach. The loop is closed in 2 ways:

A PID controller whose goal is solely to follow the position trajectory is used in parallel. The actual voltage command is the sum of the two contributions. (PID + FF) If the PID command is strongly opposite to the model-based command, the second will be ignored until their contributions are compatible. To avoid instability injections with this method, an hysteresis is used to decide the switching ON or OFF of the model contribution.

The user can take corrective actions in the next trajectory (150 ms later) and even choose to stop the current trajectory playing at any time.

3 Models

3.1 DC Motor

The chosen model for the Direct Current (DC) motor has only 2 parameters and expresses the classical linear trade-off between speed and torque at a given voltage U. R is the terminal resistance and k_e is the back-EMF constant

$$U = \tau \times \frac{R}{k_e} + k_e \times w \tag{3}$$

3.2 Friction

In [6] it was shown that the model (Fig. 1) was a good fit for geared transmissions at low speeds but failed at high speeds. The non-linearity between friction torque and speed that is responsible for the mismatch was not observed in the speed range of the motor, probably because the maximum speed of the motor (410 deg/s) is low enough to avoid it. Therefore, the classical Stribeck effect based friction model was chosen. In order to simplify the model and reduce its computational impact on the micro-controller, the parameter δ was fixed to 1 in Eq. 4. The same simplification was done by [3] (Fig. 2).

$$\tau_f = k_{vis} \times \omega \; - \; sign(\omega) \times (\beta \times \tau_s + (1 - \beta) \times \tau_{cc}) \qquad \beta = e^{-|\frac{\omega}{\omega_{lin}}|^{\delta}} \tag{4}$$

where ω_{lin} is the parameter defining the area of influence of the Stribeck effect, k_{vis} is the viscous friction constant and τ_{cc} satisfies: $\tau_f(\omega_{lin}) = \tau_c$.

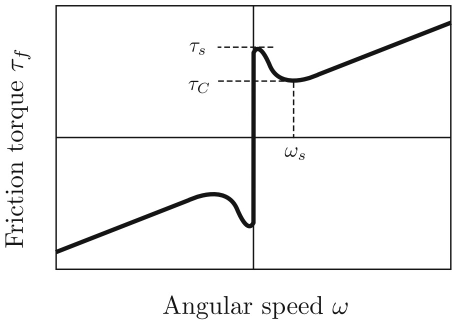

Fig. 1. Classic friction model

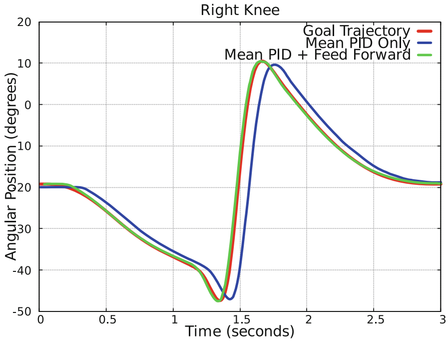

Fig. 2. Average right knee trajectory

4 Model Fitting

In order to quantify the validity of the actuator models and determine its parameters the following steps are followed:

A set of 113 measures of 1 s each are performed on a MX-64 servo-motor with nothing attached to its shaft. Predefined voltage command trajectories are implanted in the firmware and are followed and recorded with a 0.1 ms precision hardware timer. Three types of command trajectories are used, flat step commands (in order to have constant speeds), saw-tooth patterns and brutal step changes (in order to create important accelerations).

The exact same measures are recorded again on the same MX-64 with a load of approximately 0.28 kg at 12 cm.

A simple motor simulator is implemented using the electrical model and the friction model. The simulator is used to replay the voltage commands and estimate the motor position. A Root Mean Square (RMS) error is then computed for each one of the 226 measures. Therefore, a fitness function is computed. The function takes 7 model parameters (k_e, r, k_{vis}, τ_s, τ_{cc}, w_{lin}, I_0) and returns a score quantifying the quality of the model relative to a set of measures.

The CMA-ES python implementation is then used in order to maximize the fitness function. A previously hand-tuned set of model parameters is used as the starting point. Activating the restart strategy, specifying a standard deviation for each parameter and having a good starting set of model parameters are the three most important options to get the algorithm to perform well. The best match has a score 56 times better than the hand tuned solution.

5 Experimenting a Highly Dynamic Kick on Sigmaban

A dynamic kick motion is generated off-line. Standard inverse dynamics are used on a robot model to compute a torque trajectory for all degrees of freedom. The robot begins and ends the movement in single support on the left leg in a statically stable posture. The positions are sent to the motors with the classic

PID approach by updating the goal position periodically. The read/write loop frequency is 100 Hz. The position trajectories and the torque trajectories are fitted with cubic polynomials of 150 ms each and feed-forwarded to the servo-motors. The PID and the models are used by the firmware. In both cases, the PID gains are set to $P = 16$, $I = 1$, $D = 0$. The integral gain heavily reduces the static error on the starting position. 10 kicks are recorded for each method.

Note that if a control method was capable of reproducing a delayed version of the command without deforming it, complex dynamic movements could still be performed as long as the delay remains constant and identical for every degree of freedom (DoF). Therefore, a raw RMS error between the goal position and the measured position is not enough to quantify the quality of a control strategy.

For each measure and for each DoF, the measured positions are shifted along the time axis until their correlation with the goal trajectory is maximum. The shift value expresses the time delay, while the RMS error of the shifted measures expresses the deformation relatively to the ideal trajectory. The delay and the deformation are averaged over the 10 measures (Fig. 3).

The Cartesian trajectories of the right foot and the trunk are considered in a similar way.

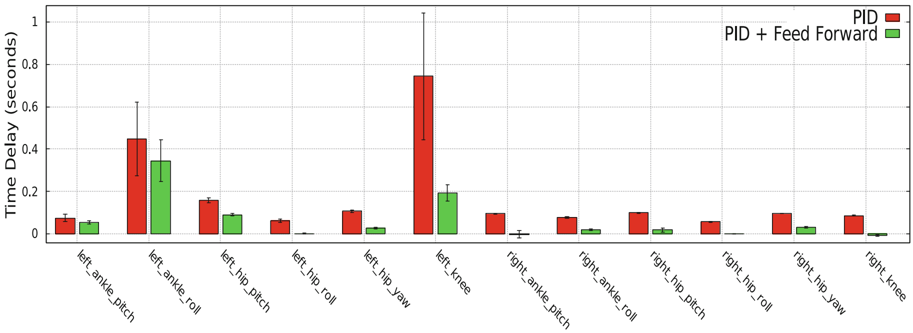

Fig. 3. Delays (and standard deviation) for each DoF and control strategy

The delay appears to be significantly reduced when the feed-forward control is used. The right_knee, which is the DoF performing the most dynamic trajectory, has an average delay of 86 ms with the PID only approach whereas its average delay is −8 ms with the feed-forward approach. The average delay for all DoFs with the PID only approach is 91 ms against 23 ms with the feed-forward control.

Note that the DoFs left_ankle_roll and left_knee have almost flat position trajectories during the kick, therefore their calculated delay is very noise dependent and will be ignored in the results interpretation. This is consistent with the fact that the delay's standard deviation is several times higher with these two DoFs. Also, note that a negative delay can be measured when the feed-forward overcompensates the command (Figs. 4 and 5).

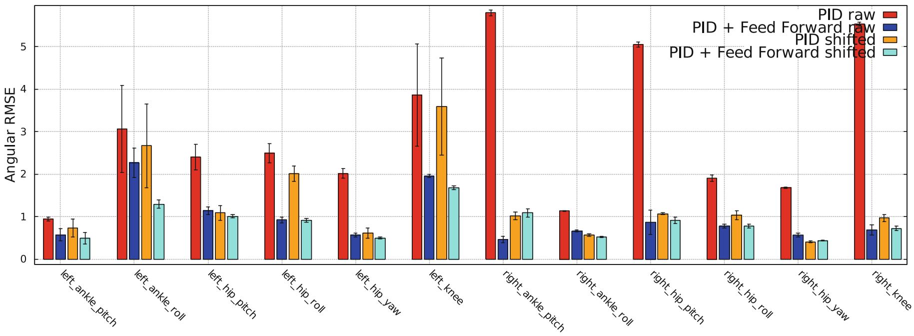

Fig. 4. RMS error (in degrees) (and standard deviations) for each DoF and control strategy, with and without time shifting

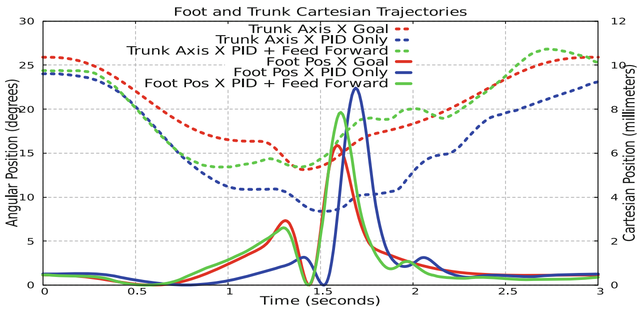

Fig. 5. Average trajectories in Cartesian space of the trunk orientation and kicking foot position for each control strategy

Once the measures are time shifted, the RMS error gap between the two approaches reduces. The first strategy (PI) has an average deformation error value of 0.95 against 0.74 for the second (PI + FF). Moreover, while the standard deviation of the deformation is comparable between the 2 approaches for low torque DoFs (kicking right leg servos), it is 3 times higher with the PID approach for high torque DoFs (supporting left leg servos). Therefore, the model-based control offers a better repeatability, especially when the applied torques are high.

When considering the Cartesian trajectory of the right foot and of the trunk, the average delay found for the PID approach is 80 ms against 0 ms with the feed-forward (here, the delay was found by minimizing the RMS while time-shifting the curve with a step of 5 ms). The RMS error of the shifted PID trajectory is 46% higher than the RMS error of the second method (0.57 against 0.39). This value represents the deformation of the overall kick motion.

The enhancements in terms of deformation were expected. A PI controller creates a non-constant delay, which increases when high speed variations and high torques are applied, thus deforming the output trajectory. In contrast, the feed-forward approach preemptively sends a command to achieve the needed speed to follow the trajectory while compensating for output torques, inertia and friction.

Notice that the whole experiment could have benefited from an external ground-truth regarding the torque and position measures. The remaining errors can be explained as being the sum of the models imperfections, both by the model itself not matching the reality and the model parameters identification. Specifically, the temperature's influence on the system was not included and the physical model of the robot was approximate. Moreover, the DC motor manufacturer states[8] that the tolerance of the motor's characteristics can reach $+/-10\%$, important hardware discrepancies between servo-motors were indeed measured. Therefore, an improvement of the presented method would be to find a set of model parameters for each servo-motor. Finally, the backlash was not accounted for. While most of the backlash comes from the gearbox and is detected by the encoder, the short and brutal modifications of friction created by the backlash are not handled. A backlash hysteresis model [5] could be tried in future work.

6 Conclusion

The open source firmware for the MX-64 servo-motor presented in this paper is meant to be a tool offering a complete control over the actuator. A simple electric DC motor model and a traditional friction model are embedded into the firmware. The parameter identification method, which uses a black box optimization algorithm and a wide set of generic measures, proved to be much more accurate than what was achieved through specific measures and hand tuning. A combined feed-forward and feed-back implementation is tested on the humanoid robot Sigmaban with a highly dynamic football kick whose inverse dynamics is calculated. When compared to a PI controller, the proposed approach absorbs almost all the delay, significantly reduces the deformation and enhances the repeatability of heavy torque movements.

Acknowledgment. The authors would like to thank Stéphane Ygorra for his help and guidance in the project.

References

1. Fabre, R., Gimbert, H., Gondry, L., Hofer, L., Ly, O., N'Guyen, S., Passault, G., Rouxel, Q.: Rhoban football club team - description paper (2016)
2. Pan, M.-C., Lin, Y.F.: Further exploration of Vold-Kalman-filtering order tracking with shaft-speed information-ii: engineering applications. Mech. Syst. Sig. Process. **20**(6), 1410–1428 (2006)

[8] Motor data and operating ranges of maxon DC motors: http://www.maxonmotor.com/academy.

3. Schwarz, M., Behnke, S.: Compliant robot behavior using servo actuator models identified by iterative learning control. In: Behnke, S., Veloso, M., Visser, A., Xiong, R. (eds.) RoboCup 2013. LNCS, vol. 8371, pp. 207–218. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44468-9_19
4. Mensink, A.: Characterization and modeling of a dynamixel servo. In: Technical report, University of Twente (2008)
5. Ahmad, N.J., Khorrami, F.: Adaptive control of systems with backlash hysteresis at the input. In: American Control Conference (1999)
6. Waiboer, R., Aarts, R., Jonker, B.: Velocity dependence of joint friction in robotic manipulators with gear transmissions. In: ECCOMAS Thematic Conference Multibody Dynamics (2005)
7. Wojtusch, J.: Development of electronics and control for servo actuators in robotic applications. In: Diploma Thesis (2011)