Development of a Series Elastic Actuator and a Distributed Computational Platform for Robotics

GONÇALO PATRÍCIO LUÍS

gpl@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal
November 2015

Abstract

Robotics embraces a wide spectrum of engineering areas. From electronics to control and artificial intelligence, passing through mechanics, robotics is the culmination of what is best on those areas. The project this thesis develops has as main goal the exploration and implementation of advanced control algorithms for making humanoid robots walk, run, jump or complete any other task not only in computer simulations but also on a small real humanoid robot. This thesis develops a computational platform where the control systems will be implemented. The developed platform uses parallel and distributed programming techniques to achieve the best performance and thus provide more possibilities to whom will implement the said controllers. In this thesis a series elastic actuator is also developed based on an AX-12 Dynamixel servo from Robotis.

I. INTRODUCTION

Since 2007 that MSc students have been working on a cheap, walking, running and jumping humanoid robot [13] [9] [8]. This thesis work developed that work further, the primary goal was to modify the robot’s actuators that work in position feedback and are high impedance, to compliant force actuators. Having force control in a humanoid robot opens the door to numerous theoretical experiments that have been successful in computer simulations but never made it to real robots. As many articles verify [7] [2] [5] [1] [4], force control is the ultimate way to control a robot that walks, interacts with humans, or handles delicate environments.

The need for new computational hardware existed because newer platforms allow more computational speed, faster communication between boards, and better compatibility with new hardware and operating systems, which results in a broader range of control systems that can be implemented. The low level software had to be re-done because there were issues in the previous version, it had serious time keeping problems and the code was generally written in a way that is far from optimal for control systems.

Structure of this article: in the second section the new hardware is addressed, in the third section the new control software is analysed, and in the fourth section the series elastic actuator is analysed mathematically and the practical solution is presented.

II. NEW PLATFORM FOR HIGH LEVEL CONTROL

From the enormous amount of possibilities, choosing a the best board is not easy, specially taking into account that all these products are development boards without much support and advertised as non finished products. The problem with this is that we can’t trust a board maker simply because they sell the one with best features, because if there aren’t any serial port drivers we can’t do anything with it
on this project. So the "trick" is to select the board with best features from the most popular boards. The mandatory feature was to have support for high speed UART communication (more than 1Mbps) to allow direct communication with the servos thus removing the need for the CM5. The selected platform was the Odroid U3 from Hardkernel. It makes use of a Samsung Exynos 4412, a quad-core ARM SOC running at 1.7GHz that costs around 60 euros.

.1 Actuator Hardware

To make a series elastic actuator we need to have two position sensors, one on each side of the elastic member on the shaft. The sensor used by the AX-12 servos is the analog potentiometer in figure [1]. The schematic on the AX-12’s allowed us to connect a second sensor in pin 24, the second sensor was taken from a burned donor servo.

![Figure 1: The Murata SV01A103 potentiometer used on the AX-12 Dynamixel servos](image)

The potentiometer used on the AX-12 servos is only capable of reading 300 degrees of rotation, so when implementing the second sensor care has to be taken to avoid increasing the force sensing deadzone by intersecting the deadzone of one sensor with the sensible zone of the other, in other words the potentiometers should be aligned.

Although the ATMega8 has a regular UART port with TX and RX lines, those aren’t readily available on the servo connector. In the connectors only 3 lines are available: power, ground, and communication. The line driver acts as a ‘Y’ and the RX/TX selection is done by two lines connected to the ATMega on pins 10 and 11. Switching the state of these lines makes the line driver put the data being received from the outside on the RX pin of the ATMega, or sending out what the ATMega is putting on the TX line, both can’t happen at the same time and this has to be taken care of on software. This chip is needed on these servos to allow for the daisy chain communication have a large number of servos connected to one another without loosing signal quality. Because of this the Odroid also needed a chip that acted as a ‘Y’ to connect the only communication wire coming from the servos to the TX and RX lines on the Odroid, the solution is presented on a latter section. Note that although these chips are fast enough to transmit data at 1Mbps, it takes some time to switch between the sending and receiving states and some time is lost here on the whole cycle of each sample time step.

.2 Communication Between the Odroid and the AX-12’s

As just mentioned the Dynamixel servos make use of a quad-buffer/line driver that acts as a ‘Y’ connector between both RX and TX lines on the MCU and the pin available on the connector, thus the Odroid also needs a similar component otherwise communication with the servos wouldn’t be possible. Another problem is that the voltage level on the servo’s UART ports is 0-5V and on the Odroid is 0-1.8V, so an aditional voltage converter is also needed. With both these problems in hand an integrated solution was designed. A PCB was built with 2 components, the voltage converter and the quad-buffer/line driver.

The quad-buffer/line driver was the same used by the dynamixels, the NXP 74HC126. The other component in the additional board is the voltage conversion module. The voltage conversion between 1.8V and 5V is done between the Odroid and the quad-buffer/line driver, so that the latter works at 5V. This meant that 4 lines had to be converted, both TX and RX, and both enable pins. The final solution was to use the bi-directional logic level converter from sparkfun because it’s an easy to use plug and play component without the need of further components.
Figure 2: Quad-buffer/line driver and voltage level converter for the Odroid

The addition of this board does not make the communication slower in any way, the components used are rated for SPI or I2C communication. What may add some latency to the communication is the time the quad-buffer/line driver needs to change state, but that component was already present on the servos.

Although both the ATMega8’s and the Odroid’s UART hardware are capable of full-duplex communication, that is not possible because of the presence of the quad-buffer/line driver as the communication direction must be selected before any messages are sent. So the only option to have full-duplex communication would either be to not have the quad-buffer/line driver and probably decrement the communication reliability on long daisy chains, or use another independent piece of hardware (SPI for instance) in conjunction with the already used UART port, and use one protocol to communicate from the Odroid to the servos, and the other from the servos to the Odroid. This idea wasn’t implemented because neither the I2C or the SPI ports are available in the servo connector.

III. Software

3 Problems with the old software

The software present on the Roboard, CM5, and the servos was very poorly optimized, focused on communicating as much as possible to maintain the data updated in every servo but at the same time relied on pooling for synchronization, which is the worse method for synchronization. The protocol on the other hand is solid and well thought out, figure 3 illustrates the communication flow. As the servos are connected via a daisy chain to the high level controller, messages can’t be directed at a certain slave. To be able to get data from all the servos without having to ask each one to send data, all the servos are listening to what is being transmitted and each servo knows when it’s their turn to return their data to the high level controller. As each servo has a unique to the robot identification number, the servo with ID=1 knows it’s the first to send its data, the rest of the servos wait for this servo to finish the communication, and after that the servo with ID=2 knows it’s his turn to send data, and so on. The communication with new control commands from the high level controller on the other hand is a big message to all the servos, the message data is composed of servo ID’s and actual data. The servos know which data is their’s because it’s the data from their ID number on, and the message size for each servo on the broadcasted message is fixed (2 bytes) so that there is no way that a servo confuses data with an ID number. The communication scheme used in the new software has the same logic as this one. Note that only the communication scheme has been taken to the new software because the actual code that implemented the old software relied on pooling instead of hardware interrupts.

Figure 3: Communication flow

Another problem with the old firmware was the speed at which the Roboard could communicate with the CM5, it communicated at 115200bps without any sample time control as the code was no more than an s-function generated by matlab. As there was no synchronization, and the Roboard communicated
much slower than the servos, the messages from the servos and the Roboard would over-
lay and the CM5 would have to save the newly arrived data from the Roboard to the next servo
cycle, because the message to the servos was already being sent at the time the Roboard
started sending its message. This caused de-
lays on the control action and made the sample
time of the high level controller random within
certain limits. Limits which were acceptable
for the previous works done but may not ac-
ceptable for future work, nonetheless having
random sample time in a digital control sys-
tem is a huge mistake that can render the best
performing controller unstable.

With these problems in hand:
1) No sample time control;
2) Pooling on the communication;
3) Poorly written code with non-optimized
flows, and poor readability and comments;
4) Using the CM5 as a communication in-
terface between the Roboard and the servos
taking up more communication time;

in code that extended through thousands of
lines, the decision to make everything from
scratch came up and the new software frame-
work was built with fresh hardware. I could
probably make the things that worked previ-
ously work again for me, but if during the
development of my thesis I needed something
a little different that required to change the
code, it would be easier to rebuild the whole
thing so, might as well do it right from the
start.

4 Pipelined Parallel Control System

The system to be controlled is a robot with up
to 18 DOF’s(12 for both legs and 6 for the arms),
each DOF’s control action will most certainly
depend on the control actions of other DOF’s
not only in the forward or inverse kinematics
fashion but also because inertial forces gener-
ated for instance in the arm of the robot will
influence the force done on the foot. Also walk-
ing methodologies like the ZMP [8] method re-
quire coupled calculation of the control actions
for all the joints. This part of the control sys-
tem requires data from all the sensors and the
calculations can’t be parallelized. But normally
these central control systems only calculate the
trajectory of the joints, leaving the control of
the actuator out. The control of the actuator is
responsible for generating the required control
action on the actuator itself such that the high
level control order is followed, normally a PID
is used for this, and contrary to the trajectory
planning control, this control system is inde-
pendent for each actuator. This means that the
low level control of the actuators can be par-
allelized. With this in hand the layout of the
computational platform for the control system
can be developed.

Increasing the sample rate gives more pos-
sibilities, The MIT Cheetah [12] makes use of
a highly optimized computational platform to
allow the required sample time to make the
robot run(4000Hz). There are no specific re-
quirements for the robot in this thesis, so no
required sample-time can be calculated, but
having the fastest possible platform from the
start is a good way to start, and when the limit
of this platform is achieved, a whole new plat-
form with faster hardware should be devel-
oped, or even a more serious robot. This is
preferred to building an “enough for now” plat-
form that may be rendered useless in less than
a year. With this perspective, the fastest possi-
ble controller platform was developed.

5 Paralelizing tasks

Since we have multiple cores at our disposal
and some of the required tasks don’t need to be
done serially, parallel processing can be imple-
mented [12] to increase the max sampling fre-
quency the system can achieve. On the Odroid
side parallelization can only be implemented in
a pipeline fashion because the tasks of the
high level controller are all dependent and thus
need to be accomplished serially. Because con-
trol calculations can only be done after the data
from the servos is received, and sending data
to the servos is only possible after the calcula-
tions have completed. Nonetheless these tasks
are separable and uncoupled, and while the control calculations from sample moment $t$ are being done, the Odroid can be receiving data from sample moment $t+1$, so that when the calculations from moment $t$ are complete, data for the next calculations is ready.

.6 Low Level Actuator Software

The easiest tasks to parallelize are the independent ones, in this case everything that happens in the servos: sensor reading, actuation, low level control calculations and communication. So instead of having each servo act in its own time window, all servos can be commanded to act simultaneously since all the servos have the same tasks and they all take the same time to complete.

One important aspect of distributed control systems is synchronization between all the sensors and actuators. The high level controller must receive the data from all the sensors from the same sampling moment, if not the control system can become unstable. Conversely the actuation must also be synchronized, servos actuation must be simultaneous from the same control action. To achieve this an independent clock was implemented that synchronizes all systems in the control infrastructure, that is analysed in the next section.

.7 System Timer

The system timer was implemented in an ATMega8 from a dynamixel servo. The timer is a program that uses a hardware clock to generate pulses, those pulses are fed to an output pin of the chip and fed into the interrupt input pins of all the servos. This way the sample time can’t drift from servo to servo. The clock wasn’t implemented in the high level controller because it runs a non hard real-time operating system, and thus has jitter. The timer is controlled by the high level controller so that the control system can be started and stopped easily since the high level controller runs on a linux machine that is easily accessible through ssh. When the control system is idle and there are no interrupts generated by the timer, the servos stay in a sleep state. All the functionality of the low level controller (the schedule analysed in the previous section) is an interrupt service routine that responds to the timer pulse. All the servos are programmed this way so that the timing requirements are met.

.8 Control System Scheduling

With the analysis of the possibilities to make the best control platform the software was created and the scheduling it follows is that of figure 4.

![Figure 4: Final system scheduling](image)

IV. Force Actuator

Robot actuators are still generally position controlled, this control method is very good for precise repetitive tasks like painting a car or spot welding. But if a robot is to be in contact with new environments or different tasks are required, position precision is not very helpful. Force control is required for safe use of robots that interact with people, the ability to limit the applied force is very important to keep the robots safe in delicate environments [2], picking up objects without damaging or dropping them is another task that requires force control [5][11]. Last but most important for this thesis, force control is important to make robots walk, or at least very helpful and brings numerous advantages [4][7]. It’s been shown that humans change the impedance of the legs depending on the walking speed and the floor surface type [6]. This brings up the concept of variable impedance.

Impedance is a measure of how much something resists motion when subjected to a force.
Conversely, compliance is how much something accepts motion when subjected to an external force. Human muscles have very low impedance, when we aren’t contracting a muscle it doesn’t resist any motion provoked by external forces. This behaviour is opposite to that of conventional robot actuators, not only position controlled but also force controlled. Normally stiffness (higher impedance) is an advantage to position control because it offers more precision and repetability but adding compliance to the robot’s actuators is a step towards getting all the advantages mentioned above plus those of passive walkers (namely energy saving and regeneration) [11].

.9 Adding Compliance

The solution adopted on this thesis was to build a series elastic actuator. A series elastic actuator adds a compliant element, in form of a spring, to the output of the actuator and senses the force by measuring its deflection. The biggest advantages that this solution has when compared to the other force actuators is that it can be built in a compact space, the controller gains can be higher because the open loop now has lower gain because the spring was added [10], and it has hardware compliance that helps controlling impact forces more easily.

.10 Series Elastic Actuators

Series elastic actuators take standard high stiffness position actuators, and transform them into compliant force actuators. In this thesis specifically cheap off the shelf servo motors (Dynamixel AX-12) are used, as the goal is to add force control and variable impedance to an existing humanoid robot (Robotis Bioloid) that uses those servos.

While stiff force sensors measure force with extensometers which require large and expensive signal processing hardware, in SEAs the force can be measured by measuring the deflection of the spring with simple position sensors by using eq. [1], where \( k \) is the spring stiffness. This is possible because the deflections of the spring are considerable and the spring can be chosen such that the minimum measurable angle of the position sensors used (resolution of the position sensors) is less or equal than the equivalent to the minimum force required to be read (required force resolution). This characteristic alone makes the SEAs the best option for this project because of the compact size and low price.

\[
F = k(x_1 - x_2) \tag{1}
\]

Analysing the actuator with the real motor dynamics is fundamental to understand the capabilities of the SEA built around the AX-12 platform, so a parametric model identified from the real servo was used. The variables that affect the performance of the SEA the most are motor performance (maximum speed, torque, and acceleration), the sample rate, and the spring stiffness. Stepping back and looking at these variables from a systemic point of view, in this project the stock motor is used so increasing the performance of the motor in the servos isn’t an option in this thesis, the hardware and software of the controllers has been optimized and is now performance capped by the hardware used and the communication protocol speed, the spring stiffness is adjustable but also has limits. The spring stiffness on the real case is bounded, the spring can’t be too stiff has to not allow the position sensor to read the minimum force that has to be read, and it can’t be too soft so that the force application speed is too slow.

The position value read by the ATmega8 has resolution of 0.29 degrees, so if the required minimum force reading (force resolution) is \( m_T \), the maximum spring stiffness \( K_{\text{max}} \) can be calculated using Hooke’s law (2 on the left), plugging the position resolution and the force resolution expression (2) (right) results.

\[
T = K \cdot \theta \\
K_{\text{max}} = \frac{m_T}{0.29} \tag{2}
\]

This gives an upper bound for the spring stiffness. A lower bound can also be calculated by knowing the maximum speed at which force
has to be applied. The force application speed for SEA’s in the contact force case (fixed output position) can be expressed as the first time derivative of Hooke’s law, with \( \theta_1 \) being the output position of the motor, this is expressed in equation 3 on the left. If \( T_{min} \) is the minimum force application speed and \( \dot{\theta}_{1\text{max}} \) is the maximum speed of the motor, then the minimum spring stiffness \( K_{min} \) can be calculated, and is expressed in equation 3 on the right.

\[
\dot{T} = K \cdot \dot{\theta}_1 \\
K_{min} = \frac{T_{min}}{\dot{\theta}_{1\text{max}}} \quad (3)
\]

As expected the same analysis can be done for the force acceleration and jerk, and should be done if such critical requirements are to be met.

So the spring stiffness must be selected between these two values, the exact value in that acceptable spectrum has to be selected according to the desired impact impedance and position tracking capabilities. A stiffer spring will be better for position tracking and a softer spring will make the actuator more compliant in case of impact. The influence of the spring stiffness on the performance of the actuator on those situations will be now analysed.

.11 Force Tracking

When in pure force control, the block diagram is as figure 5 shows, here the Motor block is the identified real motor dynamics which also encompasses the gearbox. The transfer function of the system is thus equation 4. In this equation the spring stiffness \( K \) and the proportional term of the PID controller are interchangeable, this means that we can select any spring and adjust the controller to have good performance, in fact systems with different springs can have the same exact transfer function. The difference is that with a softer spring the motor has to move at higher speeds to achieve the same force in the same amount of time, and it will saturate in situations the same system with stiffer spring wouldn’t. So the difference is that the non linearity will kick in more easily the softer the spring is.

\[
\frac{T}{T_d} = \frac{K \cdot \text{Motor} \cdot \text{PID}}{1 + K \cdot \text{Motor} \cdot \text{PID}} \quad (4)
\]

.12 Position Tracking

In position control, the inertia of the robot’s limb and additional objects if any are being carried are the sole responsible for the forces generated on the actuator and they must be taken into account. The block diagram of this situation is shown in figure 6. The spring mass block has the transfer function of equation 5.

\[
\frac{\theta_2}{\theta_{2d}} = \frac{K}{ms^2 + bs + K} \quad (5)
\]

The transfer function of the whole system is that of equation 6. Here different spring stiffnesses will yield different transfer functions, unlike in the pure force control case, because in the Mass block there is a spring stiffness term \( K \) in the denominator, which will appear in the denominator of the system transfer function.

\[
\frac{\theta_2}{\theta_{2d}} = \frac{\text{Mass} \cdot \text{Motor} \cdot \text{PID}}{1 + \text{Mass} \cdot \text{Motor} \cdot \text{PID}} \quad (6)
\]

A harder spring is always an advantage in position tracking, the controller can be tuned to act more aggressively and the system will be faster. A lighter limb will contribute similarly, less inertia to carry will require less force, thus less displacement on the spring, and ultimately better position tracking. Figure 7 shows the response of two servos, one with a softer spring.
than the other, to a unit step. The response in red is from the system with stiffer spring.

**Figure 7:** Comparison of soft and stiff springs in position tracking

The bode diagram of the system with stock Matlab tuneed controllers with soft and stiff springs is shown in figure 8. The blue line corresponds to the system with a spring of $0.16 \text{ N} \cdot \text{m rad}^{-1}$ while the red corresponds to the system with a spring of $3.45 \text{ N} \cdot \text{m rad}^{-1}$. The system with the soft spring has less bandwidth.

**Figure 8:** Frequency response of soft and stiff springs in position tracking

13 Output Impedance & Impact Tolerance

Testing the output impedance answers the question “how much force do I need to apply to move the output of the actuator to another position?” and testing the impact tolerance answers the question “How does the system react when subjected to an impact of certain energy?”.

In the output impedance case lowering the spring stiffness $K$ will lower the effect of the output position in the torque, the effect can be seen in figure 9 which is response of two systems, one with stiffer spring than the other, to the same input. A stiffer spring will also cause the motor to saturate with less abrupt position inputs than with a softer spring.

The impact tolerance was tested by subjecting the system to an impact of an object at a determined speed. To do that a simulation was done with a certain mass attached to an hypothetical arm of a certain length, with non-zero initial velocity. To analyse the response of the system the same mass with the same initial speed was tested on systems with different spring stiffnesses. The force responses are shown in figure 10, the blue line corresponds to the response of the system with softer spring. As expected the forces generated were smaller.
Conclusions

Selecting the spring stiffness for this series elastic actuator requires knowing the requirements for force resolution, force application speed, output impedance, impact tolerance and position tracking performance. The acceptable values for the spring stiffness are bound from the start by the force resolution and force application speed. The performance of the actuator in position tracking, impact tolerance/output impedance and force tracking has to be analysed as a stiffer spring is advantageous for force and position tracking, but a softer spring is advantageous for impact tolerance/output impedance. If actuators with different spring stiffnesses are controlled by the same high level controller, the sampling rate must be selected as to be acceptable by all the actuators, because different spring stiffnesses require different sampling rates.

Building a new all metal shaft

Having reliable and trustworthy hardware is crucial for the development of project of this nature. If the actuators aren’t precise and/or require much maintenance, progress is delayed because time is being consumed in solving actuator problems instead of in developing better control strategies. With this in mind a solid solution was needed for the future progress of this project so that the next person will be able to start his/her contribution with all the actuator problems solved. The final solution is an all metal machined shaft composed of two pieces that fit together without the need of tools or a bonding agent. The design of the new shaft allows easy installation of position sensors and its flexibility can be adjusted by varying the size of the cross part. The design has been created and tested with a CAD design tool, figure 11 shows an FEM analysis of the shaft.

References


