Örebro University

School of Science and Technology, AASS

Control of The Velvet Fingers

– Internship final report –

Author:
Kwieciński Krzysztof

Supervisor:
Dr. Robert Krug

September 22, 2015

Contents

Abstract

In this paper different approaches to control of a gripper are presented. The main focus is on the current (force) control. A gripper used for experiments is The Velvet Fingers, a simple tactile under-actuated gripper.

Figure 1: The Velvet Fingers

1 Introduction

The report treats of effects of my internship that took place at *School of Science and Technology* at the *Örebro University* in Sweden. I worked for *Centre for Applied Autonomous Sensor Systems*, at the *Mobile Robotics and Olfaction Lab*. A time span I spent here was two months, namely August and September 2015. My main goal was to test different ways of control The Velvet Fingers. Since the gripper is a smart gripper using a novel concept of the end-effector combining simple under-actuated mechanics with high manipulation possibilities, there is only need to control the DC motors.

2 Analysis

2.1 DC Armature Model

A DC motor can be described by two equations:

$$V(t) = L\frac{di(t)}{dt} + Ri(t) + E(t) \tag{1}$$

$$J\dot{\omega}(t) = k_{tau}i(t) - \mu\omega(t) - T_L(t), \tag{2}$$

where 1 describes electrical and 2 describes mechanical aspect of the motor.

It is known that

$$E(t) = k_{emf}\omega(t), \tag{3}$$

$$T(t) = k_{tau}i(t). \tag{4}$$

Combining the above equations and assuming that changes in current are instantaneous yields

$$V(t) = \frac{R}{k_{tau}}(J\dot{\omega}(t) + \mu\omega(t) + T_L(t)) + k_{emf}\omega(t) \tag{5}$$

or simply

$$V(t) = Ri(t) + k_{emf}\omega(t) \tag{6}$$

if the current is measured.

Equations 5 and 6 hold for every state of the motor. However, since we want the gripper to hold an object firmly, desired ω and $\dot{\omega}$ are zero. That means the final feedforward equation when holding an object can be reduced to

$$V(t) = R\frac{T_L(t)}{k_{tau}} = Ri(t). \tag{7}$$

Nomenclature: R - motor terminal resistance, L - motor terminal inductance, i - current, E - back electromotive force, J - rotor inertia moment, ω - rotor angular velocity, $\dot{\omega}$ - rotor angular acceleration, k_{tau} - torque constant, k_{emf} - speed constant, T_L - load torque.

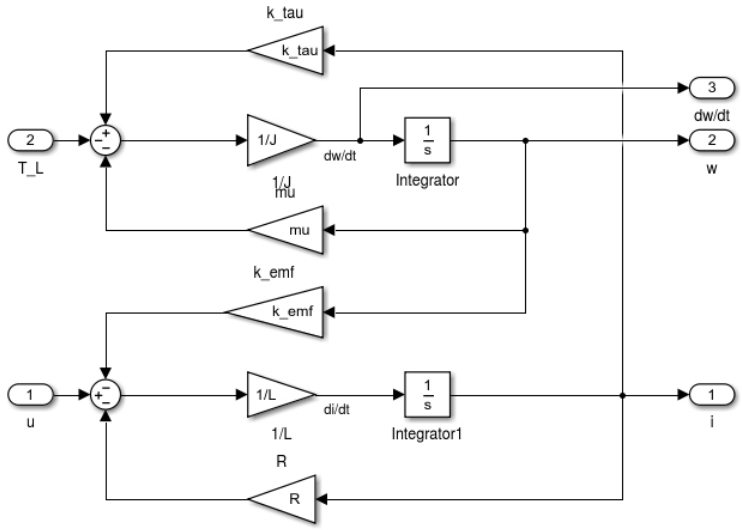

Figure 2: A model of a DC Motor in Simulink built of basic blocks

3 Implementation

3.1 Matlab

The first step was to understand the working principle of a DC motor. After getting such knowledge, I had to verify it in Matlab and Simulink. I created two models of a DC motor in Simulink: one built of only basic blocks and second using blocks from the Simscape library. Models are shown on figures 2 and 3.

I tested both models using scripts written in Matlab. Inputs of the system were voltage and applied force, and outputs were current and angular velocity and acceleration. Tests were done for open and closed loop. Control of the process variable was done with a PID controller. Results for both models were almost identical and the simulations confirmed theoretical knowledge and developed my intuition.

3.2 Arduino

Afer getting some experience, I started working with hardware. The computations and control were done by Arduino Due board. The code was written in C++. With such combination, I was able to write a program containing a lot of operations using a high-level programming language.

Control of speed and direction of the motor's movement was done with use of an external H-Bridge and a PWM signal from Arduino.

Arduino was connected with a computer through a serial port. As a middleware The Robot Operating System was used. ROS allowed to change parameters on-line and provided opportunity to observe and visualize values of chosen variables.

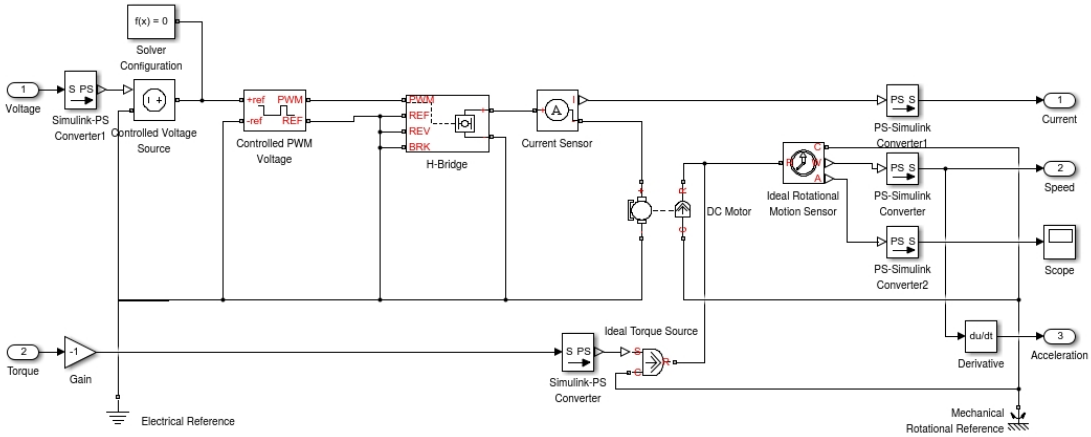

Figure 3: A model of a DC Motor in Simulink built of blocks from the Simscape library

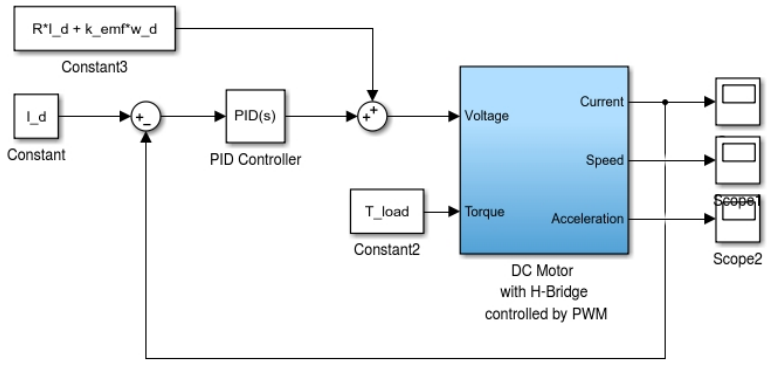

Figure 4: An example control loop

Thanks to usage of encoders and measuring the current flowing through the motor, I was able to implement various types of control, for instance:

- current (force) control,
- position control,
- velocity control,
- stiffness control,
- impedance control,
- parallel force/position control.

Because the control was dedicated for a gripper, the most important was the first one.

3.3 Documentation

A detailed documentation of the code was generated with Doxygen and is available at the GitHub repository.

4 Case studies

For conducting experiments two setups were used. A basic one consisted only from one DC motor. The second was The Velvet Fingers.

4.1 Essential methods

Three fundamental methods of control are: current, position and velocity control. They all rely on measuring one process variable (PV) and try to reach the setpoint (SP) with use of a PID controller, which output is the control variable (CV).

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de}{dt} \tag{8}$$

4.1.1 Current Control

Current control was really effective. The setpoint was reached very fast and without oscillations.

Current control can be identified with force control. Even though the explicit force is not measured, the desired force can be calculated from 4.

4.1.2 Position Control

Position control was really effective. The setpoint was reached almost perfectly and without oscillations.

4.1.3 Velocity Control

Velocity control was effective. The setpoint was reached.

4.2 Complex methods

When interacting with the environment, there is need for more sophisticated methods. The control variable is calculated based on two or more process variables.

Nomenclature: p - current position, $\dot{p}$ - current velocity, $\ddot{p}$ - current acceleration, f - current force, $_d$ - desired, K_p - position coefficient, K_v - velocity coefficient, K_f - force coefficient, K_i - integral coefficient, M - mass, B - damping, K - stiffness.

4.2.1 Stiffness Control

Stiffness control derives from a position control scheme of PD type.

$$u(t) = K_p(p_d - p) - K_v\dot{p}, \tag{9}$$

4.2.2 Impedance Control

In impedance control the input is flow (velocity or displacement) and the output is effort (force). In directions where the robot has to be environment-sensitive, impedance is low, otherwise impedance is high. The desired impedance behavior is a trade-off between trajectory error and force error. If the motion is unconstrained force will be zero and the robot will move to the reference position.

$$u(t) = M(\ddot{p}_d - \ddot{p}) + B(\dot{p}_d - \dot{p}) + K(p_d - p) - K_f f \tag{10}$$

Desired acceleration and velocity are usually not present to guarantee passivity when in contact with the environment.

4.2.3 Parallel Force/Position Control

In order to combine the features of stiffness control and force control, a parallel force/-position regulator can be designed where a PI force control action plus desired force feedforward is used in parallel to a PD position control action. The position controller is an impedance while the force controller results in a filtering action on the force variables.

$$u(t) = M\ddot{e}_p + K_v\dot{e}_p + K_p e_p + K_f e_f + K_i \int_0^t e_f d\tau, \tag{11}$$

where error e

$$e_{variable}(t) = variable_{desired} - variable(t). \tag{12}$$

5 Conclusion

All tested methods of control work as expected. Working with hardware might be a source of problems that are different to track down. Really important is filtering variables, i.e. current and position. Tuning PID plays a crucial role in control. If properly tuned, the controller is fast and precise. Achieving two different goals is impossible. The point is to find a fair trade-off between them.

References

[1] J. Fraden. *Handbook of Modern Sensors*. Springer, 2010.

[2] K. Kwieciński. Stworzenie modelu wybranego robota w programie gazebo (założenia projektowe), marzec 2015.

[3] Advanced Navigation. `http://www.advancednavigation.com.au/product/orientus`, kwiecień 2015.

[4] Gazebo Sim. `https://www.youtube.com/channel/UCJyqf9XJpDoM9XnpAwW6WxA/`, marzec 2015.

[5] Global Land Cover Facility. `http://glcf.umd.edu/`, maj 2015.

[6] Hokuyo. `https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html`, kwiecień 2015.

[7] Microsoft. `https://www.microsoft.com/en-us/kinectforwindows/meetkinect/default.aspx`, kwiecień 2015.

[8] Open Source Robotics Foundation. *Gazebo*. `http://www.gazebosim.org/tutorials`, 2014.

[9] Open Source Robotics Foundation. *Gazebo API*. `http://gazebosim.org/api.html`, 2014.

[10] SDFormat. `http://sdformat.org/`, kwiecień 2015.

[11] SketchUp. `http://www.sketchup.com/`, kwiecień 2015.