Juggling Robot Project

Johnny Mieses [1,2] , Martin Trngren[2], Xinhai Zhang [2], Pouya Vahdati[2],
Lars Svensson[2], Yunjin Gu[2]

[1] Department of Mechanical Engineering, Stony Brook University, SUNY, Stony Brook, NY 11794, USA
[2] Division of Machatronics, Department of Machine Design, Royal Institute of Technology, Stockholm, Sweden

1 Abstract

This project aims to reproduce the human like behavior of juggling balls through a robot mechanism using computer vision. As per now, the project is in its early stages where an embedded control system is needed to create a control loop design. For the moment, the first prototype should be able to juggle one ball.

This project employs a brushed DC motor with an encoder as the feedback sensor. In addition, the embedded control system consists on the integration of the BeagleBone Black and the Arduino Mega 2560. The BeagleBone Black used in this project carries an AM335x processor, while the Arduino Mega carries an ATmega1280. As a result, the collaboration of these two boards consists in sending the commands through the BeagleBone Black such as PWM signal and duty cycle, and receive the encoder signaling with the Arduino Mega. The board connections serve to implement an open loop control.

2 Introduction

Juggling balls in the air is a practice that can take years to master depending on how many balls one wants to juggle. The successful juggle of balls depends on different parameters such as air resistance, gravity, and the varying forces exerted by the juggler. These characteristics should be controlled by the juggler to be able to juggle the balls. Similarly, the robot designed to juggle objects must control these parameters to successfully juggle the balls. One way to stablish a robot control is to adjust these parameters in real-time. For this, an embedded control system is essential to provide the necessary feedback response to the system. Embedded system control is a fundamental part of robotics, where it interfaces between the mechanical system and the software of the robot. In this project, the

embedded system will receive information about the ball position from a camera which will track the ball during juggling.

The focus of this research is to implement an electronic and embedded platform for control strategies of the juggling robot. For this, the idea is to use Simulink environment to implement the control system design. The basic components of the electronic setup included an Arduino Mega, a BeagleBone Black, an encoder, and a DC motor.

One of the primary goals of the embedded design was to establish a communication between the robotic system and Simulink. By this, the robotic system could be control using this Simulink graphical interface. Simulink offers various features dedicated to communication with Arduino and BeagleBone Black, although this last one is relatively new and still need more development. The encoder served as the sensor to read the motor speed and its direction of motion. This reading is achieved by storing the signal from the two channels of the encoder, then comparing these signals with the new incoming signals. This approach produces a problem, since the encoders signals were received and stored in the microcontroller device, which is sending a PWM signal to run the DC motor at the same time. Ideally, this problem could be solved by implementing an interrupt in the BeagleBone Black, which would capture the encoders signal and still be able to send the PWM signal to the DC motor. The BeagleBone Black is supposed to be fast enough for the interrupt to work without delaying the DC motor. The Arduino Mega served as a test platform to implement the interrupt running the DC motor. By manipulating registers of the Atmega 2560 chip, it was possible to implement external interrupts on the Arduino Mega to receive the encoder signals [1][2]. The interrupt worked as intended producing the step count output from the encoders reading, however the speed calculations did not produce the expected output. Furthermore, the interrupt implementation on the BeagleBone Black proved to be very complex within the time constraint of this project. The BeagleBone Black have different ways of implementing this interrupt, however accessing its registers via memory map is a more desirable approach to use it in conjunction with Simulink. Ultimately, the readings from the encoder were taken by the Arduino Mega, while the PWM signal was produced the BeagleBone Black. This approach provided some advantages, since the Arduino I/O was available for reading the encoder signals through Simulink. In contrast, one major disvantage of this approach is that the BeagleBone Black is a faster platform that in theory could process the signals faster than the Arduino Mega.

3 Embedded System Components

3.1 Brushed DC Motor

In general, brushed dc motor are widely used in industry for several applications including motor control by a microcontroller. A common feature among brushed dc motors is that all of them are built with the same basic components such as: a stator, rotor, brushes, and a commutator [3]. A DC motor RE 35 Graphite Brushes was used for the robotic

system [4]. The motor provided the rotary motion, which was converted to a linear motion through the worm gear in the mechanical system. In addition, an incremental encoder is attached to the motor for speed feedback.

3.2 Encoder

The encoder used in this project is an incremental optical encoder. This encoder contains three channels namely channel A, channel B, and channel I, including the voltage supply and ground connection. For this project, only channel A and B are of use, meaning that they can only measure relative changes in position [5]. Channel A produces a digital pulse before channel B if the motor is rotating in the clockwise direction, and the opposite is true for the counter-clockwise direction. Provided with a developing environment such as Arduino IDE, a pulse can be stored and compared with the next pulse which indicates the direction of rotation of the DC motor. Using this concept, the speed can be calculated using a timer interrupt which should interrupt the running code every 10ms.

3.3 Microcontrollers

The BeagleBone Black is a low-cost single board computer. It carries an AM335x 1GHz ARM Cortex A8 processor, and it supports USB, HDMI, and Ethernet. Simulink models offer a communication environment for the BeagleBone Black platform using Embedded Coder, and it offers a computer vision system toolbox. In addition, it is compatible with MATLAB, where it is possible to write scripts to control the pins of this board.

Figure 1: BeagleBone Black

In addition to the BeagleBone Black, an Arduino Mega was used in this project. The Arduino carries an ATmega2560. This microcontroller is wildly used for developing projects, and there is abundance information for several applications. Similarly, Simulink and MATLAB offers an interconnection with this developing platform for controlling its pins.

3.4 Motor Driver Controller

The ESCON 50/5 is the driver controller used in this project. The ESCON 50/5 is a PWM driver controller that provides control of permanent magnet-activated brushed DC motors

Figure 2: Arduino Mega

or brushless EC motors [6]. This servo controller serves to amplify the voltage from the microcontroller, in addition to other features that work in conjunction with the DC motor.

4 Experimental Design

4.1 Software Configuration

The Simulink environment was used for the reading of the encoder signals trough the Arduino Mega. This reading was achieved by including the I/O library to the Simulink package [7]. This library reads the encoder signals and allows some Simulink blocks for the signal processing of the encoder. The library is installed trough the Arduino IDE, and once install the Arduino IDE must be closed to stop any communication through its serial port besides Simulink. For the Simulink model, the connection to the Arduino Mega is made through the Arduino IO Setup block, which establishes a communication from the serial communication port in the Arduino to the port from the desktop computer.

Figure 3: Serial Communication Block

In addition, the Encoder Read block reads the encoder signals, and it offers some additional capabilities such as specifying the period of signal reading.

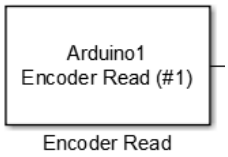

Figure 4: Encoder Reading Block

MATLAB supports the BeagleBone Black board by the installation of the Embedded

Coder Support for BeagleBone Black Hardware. Once this package is installed, it offers a variety of blocks for data acquisition from sensors and vision systems. The PWM block sets a digital pin from the BeagleBone Black as an output to send a PWM signal. It accepts a range of values from 0 to 1 to specify the duty cycle from 0% to 100%.

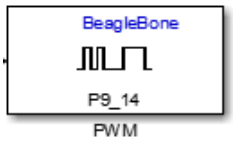

Figure 5: PWM Block

Another useful in Simulink block in this project is the GPIO Write block, which sets a pin to a physical output. This block takes Boolean input data types. Like the PWM block a pin must be selected for this block.

Figure 6: Digital Write Block

The overall Simulink model contains the above blocks. In addition, the Simulink model contains gain blocks for the motor speed calculations. By introducing these gain blocks, the speed can be obtained in rpm. Other parameters such as the sample time are part of the model.

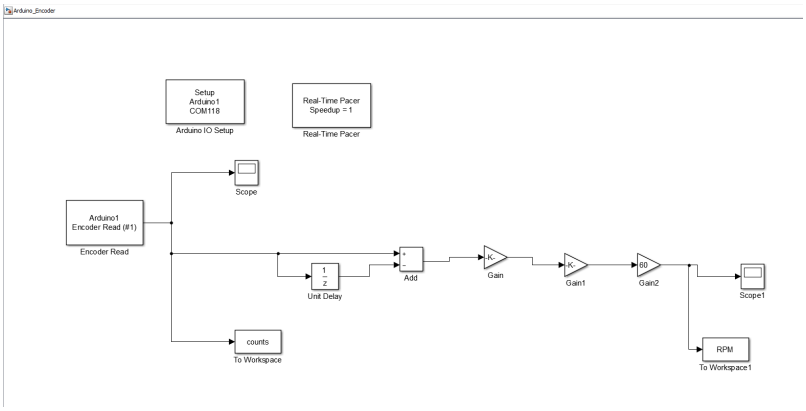

Figure 7: Digital Write Block

In addition, a MATLAB script was used for most of the testing due its simplicity. The script controlled the PWM signal from the BeagleBone Black. The script communicates to the BeagleBone Black by creating a bbb object, which includes a variety of functions for software development with this board. First, the BeagleBone Black object bbb is iniatiated, this allows the detection of the board. The writePWMFrequency function in MATLAB takes the bbb object and sets the frequency of the PWM signal. Similarly, the writePWMVoltage function establishes the output voltage at 100% duty cycle. The figure shows the communication handling of the BeagleBone board through the MATLAB tools.

```
bbb = beaglebone;

P14 = 'P9_14';
P21 = 'P9_21';
P12 = 'P9_12';

enablePWM(bbb, P14);
writePWMDutyCycle(bbb, P14, 1);
writePWMFrequency(bbb, P14, 4500);
writePWMVoltage(bbb, P14, 3);

configureDigitalPin(bbb,P21,'output');
configureDigitalPin(bbb,P12,'output');
```

Figure 8: MATLAB script showing BeagleBone object.

Two function that controlled the direction of rotation of the motor were created to be part of the MATLAB script; a function for clockwise rotation (cw) and a function for counter-clockwise rotation (ccw). These functions take three arguments namely: the BeagleBone object, the two pins that are being enable/disable on the BeagleBone Black.

4.2 Driver Controller Software Setup

The ESCON driver controller had to be setup using its manufacturer graphical software interface. The ESCON Studio 2.2 software provides a way to include the different parameters for the system to be controlled [8]. The manufacturer manual gives the necessary quantities to be included in the Escon software. For this project, it was crucial to get the maximum output from the motor to recognize the limits of the system. The setup of the ESCON software is a straight forward process, however some specific aspects are important to get the maximum output from the DC motor. The Speed Ramp option was set to No Ramp Active, so the DC motor receives the PWM signal with the required duty cycle instantaneously from the driver controller.

Another important parameter in the Escon software setup is the Maximum Output Current Limit, which as a rule of thumb, it is recommended to be two times the nominal current. Based on the ESCON manual the nominal current of the DC motor is 3.62 A, which implies that the maximum current limit should set to 7.24 A. As an open loop control was applied to the system, this was included in the ESCON software interface as well. In addition, the encoder is connected directly to the Arduino boar, so there is no need to

include a sensor parameter during the driver controller setup. The following figure shows the overall setup of the driver controller for an open loop control strategy.

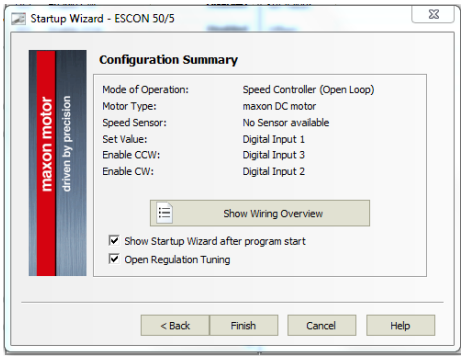

Figure 9: Overall driver controller configuration

The next step is to tune the driver controller, which implies that all the necessary connections should be made to the system. In addition, the mechanical system should be able to move freely. Supplying 48VDC to the driver controller, the following figure shows the parameters of the driver controller after the tuning process.

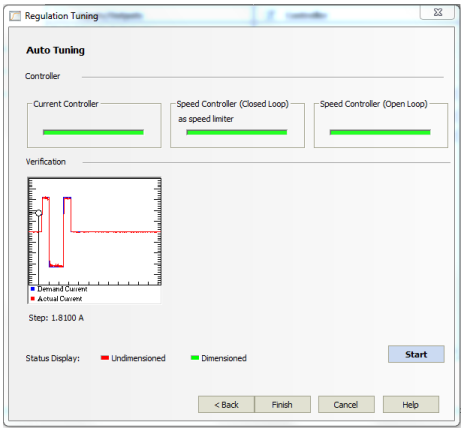

Figure 10: Sucessful tuning of the system.

5 Electronic Configuration

The BeagleBone Black is connected to the ESCON driver controller to send the PWM signal. In turn, the driver controller is connected to the motor. The board is powered through its USB when connected to a desktop computer. The BeagleBone Black uses its

pin 14, pin 21 and pin 12 to send the signals to the driver controller. The PWM signal is send trough pin 14, and the clockwise and counter-clockwise direction enabling signal are send trough the pin 21 and pin 12 respectively (depending on polarity). These pins are connected to the Digital I/O J5 header in the Escon driver controller. The connection between the BeagleBone Black and the driver controller are as follows:

$$\begin{aligned}
\text{BeagleBone Black: P9-Pin 14} &\rightarrow \text{ESCON 50/5: Digital input 1} \\
\text{BeagleBone Black: P9-Pin 12} &\rightarrow \text{ESCON 50/5: Digital input 2} \\
\text{BeagleBone Black: P9-Pin 21} &\rightarrow \text{ESCON 50/5: Digital input/output 3}
\end{aligned}$$

Similarly, the incremental encoder is connected to the Arduino Mega through its pins, and the board is powered by a desktop computer via USB. There are four wires in use from the encoder namely: channel A, channel B, voltage supply, and ground. The channel A and B are connected to the pin 2 and 3 of the Arduino respectively. The Arduino receives the signals as inputs. The pin connection is the following:

$$\begin{aligned}
\text{Encoder: Channel A} &\rightarrow \text{Arduino Mega: Digital pin 2} \\
\text{Encoder: Channel B} &\rightarrow \text{Arduino Mega: Digital pin 3} \\
\text{Encoder: VCC} &\rightarrow \text{Arduino Mega: VCC} \\
\text{Encoder: Ground} &\rightarrow \text{Arduino Mega: Ground}
\end{aligned}$$

The brushed DC motor is powered by connecting it to the pin 1 and 2 in the J2 header of the driver controller for the positive and negative polarity respectively. The encoder sits on top the DC motor so it can catch the motor rotation. The driver controller obtains 48 VDC from a power supply connected to the J1 header.

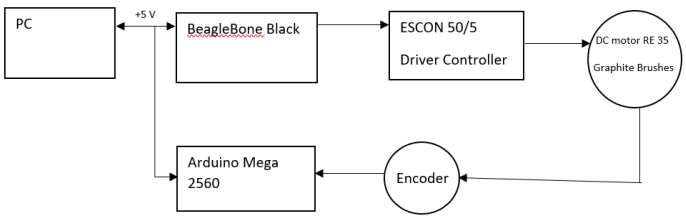

Figure 11: Overall electrical connection of the system.

6 Results and Discussion

The data of various tests were collected to present the behavior of the motor. These tests were run with a ball on the basket, and other tests were run without the ball. In addition,

the period was changed during different tests to see the impact of this parameter in the collection of the signals from the encoder. The MATLAB script for the BeagleBone Black was run in parallel with the Simulink model for the Arduino Mega. The tests were run with a PWM signal of 4.5 KHz.

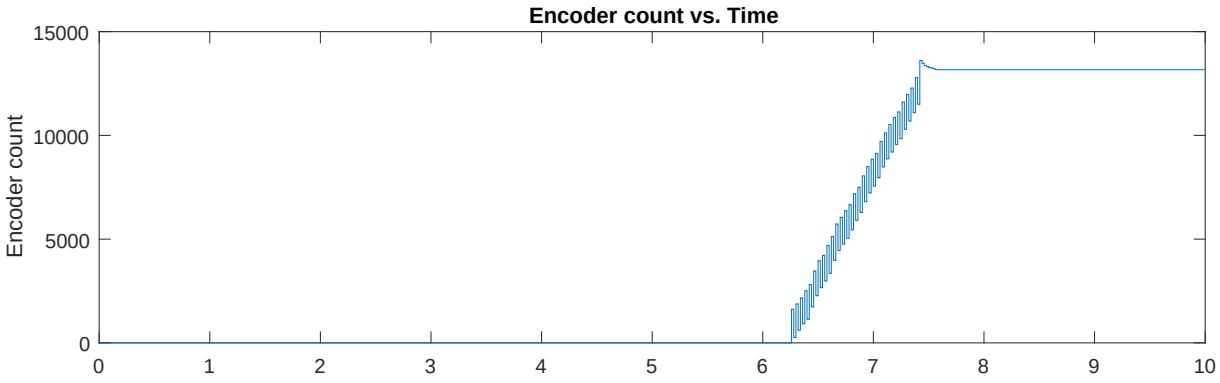

Figure 12: Encoder signals versus the sample time specified on Simulink.

The motor starts spinning after 6 seconds has past due to the time it takes the MATLAB script to run. As the motor runs, the encoder signals are added or subtracted depending on the direction of the motor. This served to count the overall steps of the signals received by the Arduino. As can be seem from the fig.11, the count is incrementing more than decrementing which means that the motor spends more time going in one direction than the other (depending on polarity). The graph is not a smooth curve, which may indicate some signals are being missed.

The speed of the motor is obtained by applying gains to the encoder block output on Simulink. The speed of the motor results from dividing the present step by the sample time, which again is multiplied by the gear ratio. Then, to obtain the speed in rpm the result in multiplied by 60. The following figure shows the result of these calculations:

Fig. 12 shows the behavior of the motors speed as it goes up and down, as specified by the cw and cww functions in the MATLAB script. At this point, the performance of the motor was sufficiently strong for putting the ball into the air. As the system was run in open loop mode, the ball kept jumping from the basket until it fell from it. The motor would keep going until the counter established in the MATLAB script reached its limit, since there was not feedback control. This speed estimate presents some noise, this could be due to encoder counts being missed. In addition, the speed of the motor is varying with time, which can make the steps count not to fall within the sample time. A simple filter was applied to the output signal to get a better estimation of the motor speed.

This filtered speed shown in fig.14 estimation shows more efficiently the picks in speed of the motor from a different test.

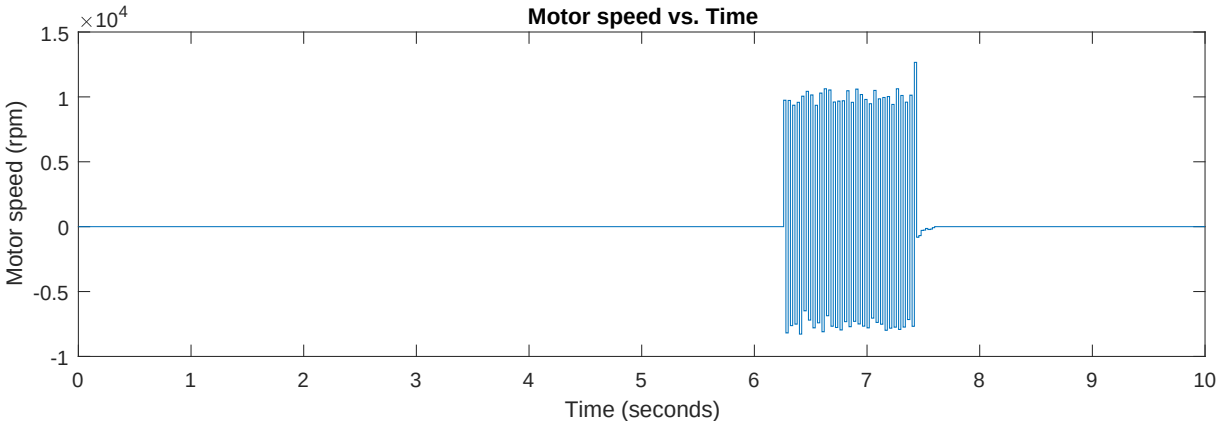

Figure 13: Motor speed versus the sample time specified on Simulink.

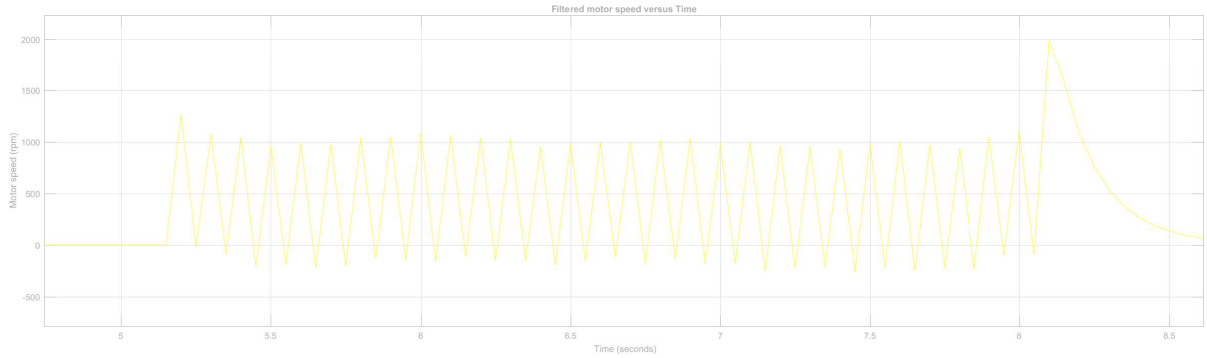

Figure 14: Filtered motor speed versus the sample time specified on Simulink.

7 Conclusions

The final setup of the electronics system worked as intended when applying an open loop control. The ball could go into the air and juggled a few times before falling off the basket due to the lack of feedback control. In addition, the distance that constrained the basket may be limiting the performance of the motor. This gap may be too small for the motor to have enough time to developed at its maximum. In that case, the mechanical system should be reviewed, and an improvement should be made to this design. If an improvement is needed, further analysis of the desired ball dynamics should be applied. Regarding the signal processing, the analysis of the system required more development, which it could not be achieved due to time constraints. This deeper analysis includes a better filtering of the output signal from the encoder, so the original signal is not missed. The Arduino Mega served to received and analyzed the encoder signal, but unfortunately, the analysis

in this project did not show if the Arduino is able or not to handle this system. However, this first prototype can be used to produce enough data to apply different changes in the future of the project.

8 References

[1] Anon, (2017). [online] Available at: http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf
[2] Interrupts, E. (2017). Efficiently Reading Quadrature With Interrupts. [online] Makeatronics.blogspot.se. Available at: http://makeatronics.blogspot.se/2013/02/efficiently-reading-quadrature-with.html
[3] Operation, P. O. F. (2004). Brushed DC Motor Fundamentals. Microchip Technology Inc., 110.
[4] Epaper.maxonmotor.com. (2017).
[5] Embeddedrelated.com. (2017). How to Estimate Encoder Velocity Without Making Stupid Mistakes: Part I - Jason Sachs. [online]
Available at: https://www.embeddedrelated.com/showarticle/158.php
[6] Reference, H. (2013). Escon 50/5, (September).
[7] Ctms.engin.umich.edu. (2017). Control Tutorials for MATLAB and Simulink - Simulink ArduinoIO Package. [online]
Available at: http://ctms.engin.umich.edu/CTMS/index.php?aux=Activities_IOpack
[8] Maxonmotorusa.com. (2017). Control Escon Page. [online]
Available at: https://www.maxonmotorusa.com/maxon/view/content/ESCON-Detailsite