Sensors and Motors Lab

Individual Lab Report #1

Paulo Camasmie

Team D – CuBI

Teammates: Jorge, Nithin, Bobby, Laavanye

February 11, 2019

Individual Progress

Sensors and Motors Control Lab

I was responsible for implementing the PID controller for the DC motor with encoder and PID. My approach was first to understand the hardware and its mechanics.

The DC motor used, had a planetary metal gear box that provided the reduction gear ratio of 50:1, and a substantial static torque, which would be a challenge for small angle position variations.

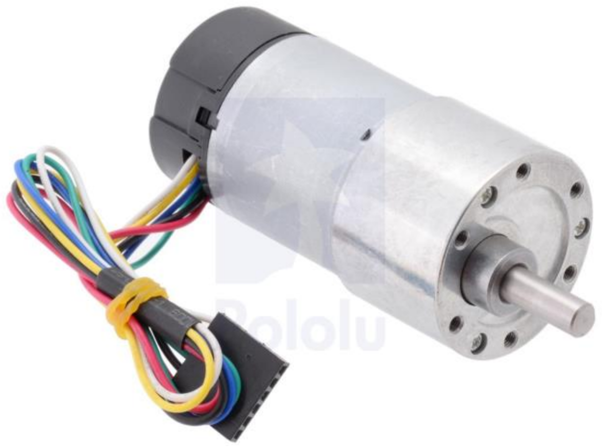


Figure 1 50:1 Metal Gearmotor 37Dx70L mm with 64 CPR Encoder
Source: www.pololu.com

The motor has an integrated quadrature encoder, providing a total of 64 pulses per revolution. For simplicity and knowing that our team had a limited number of interrupts in our microcontroller board, I decided to use only one of the encoders. The reason for using two encoders would be to determine the direction of rotation of the motor, but my approach was to command the direction of rotation and use a flag variable to be updated according to it. Since the encoder had 16 poles and the system was to read the ‘CHANGE’ signal in the interrupt, that still provided 32 counts per revolution, which considering the gear ration, translated to 1600 counts per revolution of the output shaft, which seemed like a good resolution for the feedback loop.

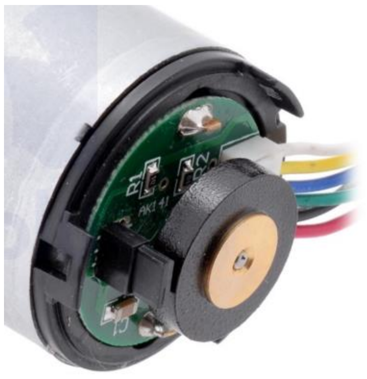


Figure 2 64 Count Per Revolution (CPR) Encoder

The motor was then driven by a Solarbotics L298 Motor kit, that incorporates the L298 dual-motor driver IC. Since it was delivered in a kit form, it was a great opportunity for our team to come together and get some soldering training from our master 'solderer', Bobby.

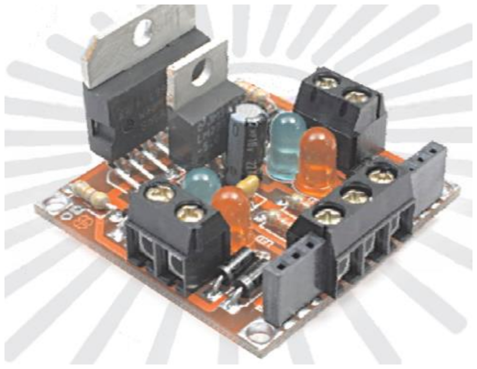


Figure 3 Solarbotics L298 Motor Driver

I believed I had a good understanding and scheme for a PID controller, but before that I was sure to make the motor run and especially being able to read the encoder pulses and translate it to the physical motor speed accurately.

The reading for the shaft position was straight forward. Since I kept a tally of a counter for the number of poles from the initial position, I wrote the following equation (ppr being poles per revolution):

$$angular\ position = \frac{pole\ count * 360}{gear\ ration * ppr\ encoder}$$

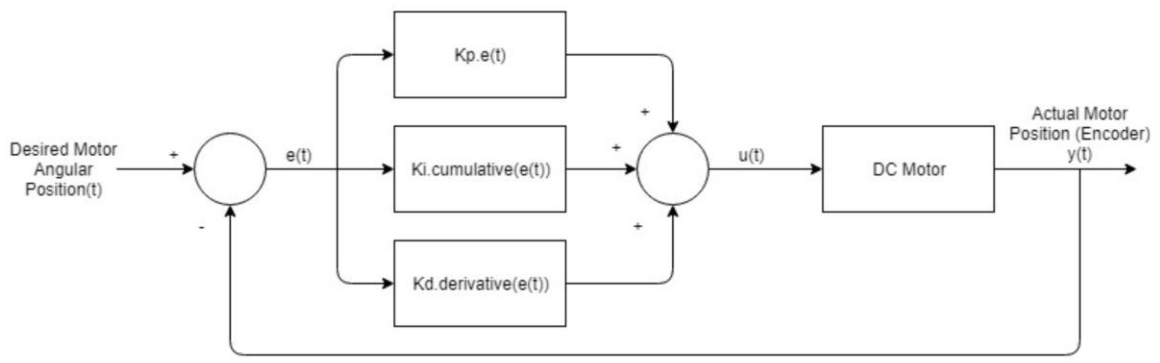
However, determining the angular speed was the biggest challenge I faced for this implementation. After many tries reading the amount of time passed between each pole count, I decided to ask Prof. John Dolan for an input. He indicated me that the time elapsed between each pole count would be too fast and inaccurate, and that I should implement a non-block timer function that would check how many poles were counted at every time interval. I did just that, implemented a function to count the number of poles every 100ms and got a consistent reading for a proxy for speed.

Having the number of poles per revolution, I was able to write the following equation for the actual angular speed, which seemed very consistent as I tested from 15 rpm up to 180rpm, which was close to the max nominal speed of 200rpm. I could not move the motor at less than 15 rpm due to the high static friction of the system mentioned above.

$$angular\ speed = \frac{delta\ poles * 60000}{gear\ ratio * ppr\ encoder * time\ interval}$$

I went then to finally implement the PID controller for position and speed. Each had a feedback loop with the target variable of interest, position and speed. The input for the controller was the error between the desired target and actual variable read at the shaft, again position or speed. The controller then applied a proportionate gain to the error between desired and actual target variable, a derivative gain to the rate of change of the error and finally an integral gain to the cumulative error. The absolute value of the sum of these components, were saturated to a max of 255 and passed to the motor as the desired motor power by the controller. A flag was switched in case the rotation of the motor flipped at that time step.

Shown below is the scheme for the angular position PID controller, which is similar in concept to the angular speed controller.



Desired Motor Angular Position(t)
+
-
e(t)
Kp.e(t)
Ki.cumulative(e(t))
Kd.derivative(e(t))
+
+
+
u(t)
DC Motor
Actual Motor Position (Encoder) y(t)

Figure 4 Position PID controller

The speed controller during test, worked very well in both directions between 15rpm to 180rpm, consistently. The gains used were fairly small. A Kp of 0.5 seemed sufficient for a short rise time, a Kd of 0.1 was sufficient to avoid any oscillations since the motor and its planetary gear seemed to be helped by its inertia and a low proportionate gain, and a very small Ki of 0.001, simply to eliminate a steady-state error.

However, ironically the position controller imposed more challenges. I had issues with smaller angles for the position controller because of a large steady state error. So my instinct was to add increase an integral gain, which seemed fine initially. But suspiciously, I let the motor get close to its goal and after 30 seconds it would go 'crazy'. During our Manipulation Estimation and Control class, Prof.George Kantor has shown us many scenarios for PID tuning and he was very adamant that the integral gain was something to thread very lightly. He gave an example of a robot being stuck on a carpet while the error would wind up and the robot when free would overshoot. Well, I was that happening with our little DC motor. I then changed the strategy, to instead of using an integral gain, to simply use three different brackets of gain, depending on the target angle, as follows:

Angles over 90 degrees: proportionate gain only Kp=0.5

Angles between 45 and 90 degrees: Kp=0.7 and Kd=0.1. In this case, I had to add a derivative control, because the increase Kp needed to take the motor out of its static position, started causing signs of overshoot.

Angles less than 45 degrees: Kp=0.9 and Kd=0.12

The implementation seemed successful, and I tested in many scenarios. More importantly, the experience was incredibly rewarding and I felt that I developed a much better understanding of actually implementing a PID controller with hardware from scratch.

I apologize, but I did not implement a sensor, since it became obvious that the DC control was a disproportionate amount of work and time consuming, and we came together as a team and Nithin was kind enough and offered to implement the force sensor, which I accepted. The force sensors is what drives the motor at the system level, the more you squeeze it, the higher the analog voltage signal is sent to the microcontroller input and the faster the motor spin. I made sure to look and completely understand the implementation of it.

However, the biggest experience was to have the one component integrated with the other systems of the project. The BIG lesson learned was that integration is a team sport and this project was a huge wake up call. What I thought was a perfect PID controller, became a less than perfect one, after it went the process of integration. In our team we divided the tasks in a way that one person in the end was responsible for integration and another one for the GUI. However, that proved both, a huge burden for the integrator and a lack in performance of the overall system.

After we presented our project, we agreed that for our robot CuBi integration, we will work much more closely and allow much more time and validation for a flawless and robust overall system.

Teamwork

Laavanye implemented the stepper motor and ultrasound circuit

Nithin derived and implemented the transfer functions to map voltage to real world physical measurements and designed the circuits for the force-resistance and servo.

Jorge was responsible for the micro-sensor, the button and the debouncing code, and the integration of all components and individual codes, and the communication protocol on the Arduino side

Bobby showed and trained us how to solder the board with professional quality. He also designed and implemented the GUI

CuBi

Laavanye is working with the vision system. Work was done on segmentation of 3D data and initial results were obtained. This will be improved and further work will be done to estimate the size of the segmented objects and validate the size of the objects of interest and see the minimum distance to camera, which will aid in camera placement.

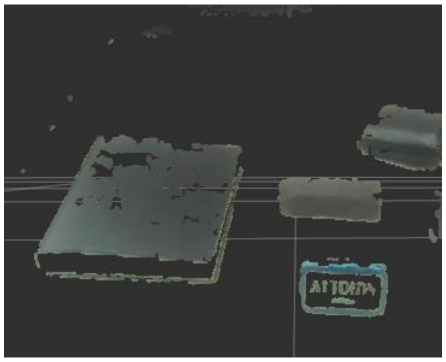


Figure 5 Object Segmentation

Jorge worked with **Nithin** on controlling the Dynamixel motors and started designing the software to interface with them at the system level. **Jorge** also focused on his on, to creating a granular schedule for our team.

Bobby has worked on the BOM and ordered all the parts needed so far for the project. He also built the Turtlebot and set it up to work with the remote control, so that we can quickly prototype many different behaviors and situations once the manipulator is ready. He also worked with Laavanye on the selection of sensors that we will be using for SLAM and grasping

Manipulator A lot of progress happened since our CoDR. My focus has been on the manipulation design and grasping strategy, since many of the aspects of CuBi will be driven by these factors. After reaching out and visiting Yifan Hou and Robert Paolini at the manipulation lab at the RI, Nithin and I discussed with them our manipulation task and opted for a 'caging" strategy for robust grasping. We were also advised to favor revolute joints rather than prismatic ones for construction simplicity and to stay away from under-actuators unless we really needed them, again for simplicity purposes. So we finished designing our first manipulator attempt, 3D printed and prototyped it and tested on samples from the Cyert Center.

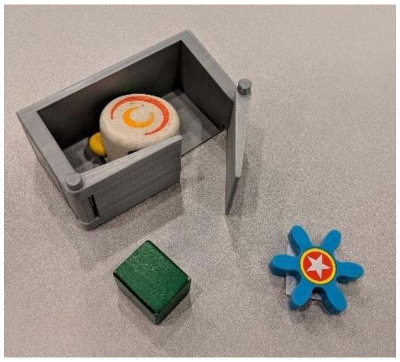


Figure 6 Cubi manipulator mockup V1

We met with our mentor, Prof.David Held who gave us a nod on the manipulation strategy and also advised to keep the system simple, to initially reach a minimum viable product and to classify objects simply by side.

Figure 7 CuBi meeting with Prof.David Held

We built a new tray prototype testing some of the ideas above and the first idea, adding a fillet to the front portion of the tray was the one the whole difference in the success rate of grabbing the objects. Below it can be seen the fillet, the offset grippers and the tapered grippers. The idea behind the tapered grippers was that they would exert a small vertical force component as they interact with objects. The ideas seems valid but there was noticeable improvement in performance and we figured that its complex shape could add cost in manufacturing. For example, instead of simply extruding the gripper profile, we would have to use a mold, forging or a 3 axis machining, and so we kept the gripper with flat side walls.

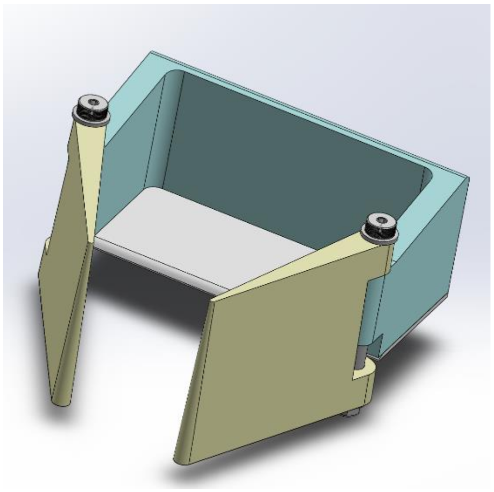


Figure 8 CuBi manipulator V2

Having approved the concept of the tray as a valid and efficient way of "scooping up" objects from the ground, both on flat surfaces and carpet, we then started designing the robotic manipulator and tray assembly. We chose to 3D print connectors to interface the Dynamixel servos on the joints, and 8020 aluminum extrusion for the longer arm members. We are also minimizing the use of Dynamixels, one per Degree of Freedom. Even Though so far we have used only Dynamixels from stock, we realize that these are very expensive parts, and we are looking ahead at the actual cost of production of the final product.

Next Steps

Below is the current concept. We should soon 3D print the desired parts, assemble the arm and control it manually. Subsequently to that, we will add the Dynamixels to replace the actual tray gripper joints and will mount the whole assembly on top of the TurtleBot mobile base we have.

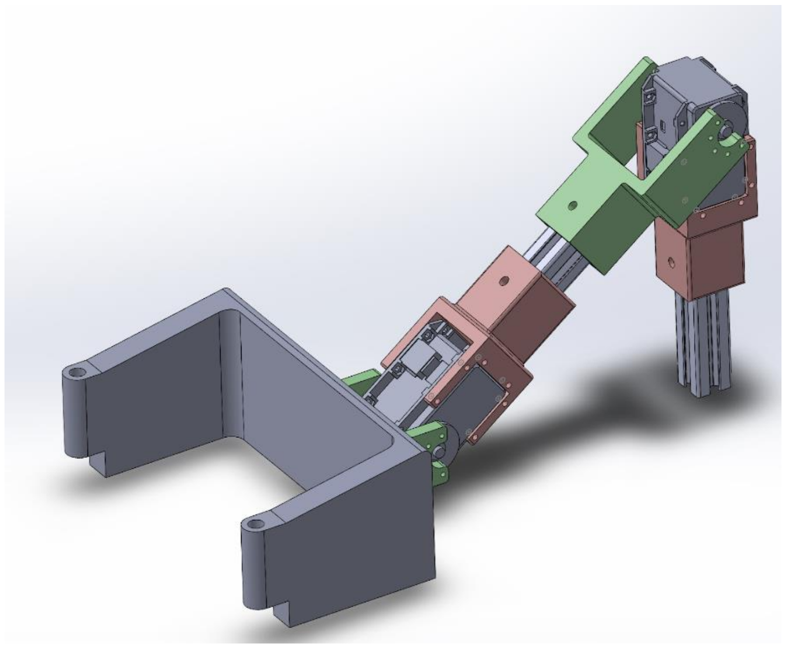


Figure 9 Front robotic manipulator with actuators

Looking ahead the challenges we have are the sensors and cameras positions, which will be mounted over the highest Dynamixel or on top of the tray. We are still keeping our options open. It would be nice to mount the sensors on top of the tray, since there is a simple transform before the tray itself and the vision, but the manipulator might be shaky. So we will put effort in having proper torque and fine tuning the controller for smoothness. Mounting the sensor at the highest point of the robot is also an option, but the tray itself might obstruct the line of sight, as it scoops up objects from the ground, especially if we decide to dump the object over the camera and on CuBi itself.

The rest of the team will be able to work more aggressively towards the integration of the vision, controls, simulation and AI as soon as I am able to deliver a mechanically working prototype. So there is a big pressure right now to deliver it on the mechanical assembly!