UNDERGRADUATE RESEARCH OPPORTUNITIES PROGRAMME

(UROP)

PROJECT REPORT

Robot End-Effector Software Development Project

Wira Azmoon Ahmad

Department of Electrical and Computer Engineering, Faculty of Engineering

National University of Singapore

Semester 1, Academic Year 2021/2022

Abstract

A prototype end-effector robot is a complex mechatronic system that requires careful design to balance the mechanical, electrical, and physical requirements. This project will study the current end-effector control software and seek to refine, test, and implement a reliable and robust endeffector control software. The end-effector has sensors, a tool spindle, and motion actuators, all of which are critical as inputs/outputs for the control software. The robot end-effector applications are aimed at light material removal (polishing, sanding, deburring) and must be able to implement force control, impedance control, and position control. The older end-effector prototypes already have working control software that can serve as a starting point, however this project will focus on implementing the control methods on a newer and smaller end-effector design.

Preface and Acknowledgements

Undergraduate Research Opportunities Program (UROP) was an excellent opportunity for me to go out of my comfort zone, and experience working with and researching technology that I was less familiar with. On top of the technical project goals, there was also the aim to gain a learning experience that I could take with me in my future endeavours. I would like to sincerely thank Dr Marcelo Ang, who provided guidance all throughout the entire semester, especially when providing advice and solutions to any issues I had faced, as well as pointing out mistakes I may have made. I would also like to thank Dr Joel Short, for introducing me to this opportunity, and being available in case I had any major issues. Last but not least, I would like to thank the previous students who had worked on this project before me, making it possible for me to build off from their successes.

Table of Contents

LIST OF FIGURES AND TABLES

Figures

Tables

1. Introduction

In robotics, a prototype end-effector refers to a flexible machine that is designed to interact with its environment. This project more specifically aims to work on a robot end-effector which does light material removal (polishing, sanding, deburring). At the same time, it must be able to implement force, impedance, and position control. Such end-effectors, used as mini robots, are already used in designs of macro-mini robots for polishing processing (Li, et al., 2020). The polishing process is widely used in various manufacturing processes and continues to require precision that continually requires novel designs to further improve (Xu, Cheung, Li, Ho, & Zhang, 2017).

This project aims to focus on implementing position control on a smaller end-effector design (Zheng, 2017), targeting the software implementation as the hardware will be fully based on previous work (Wong, 2019; Goh, 2020; Thoe, 2020; Yang, 2021). The basis of a robust control system can be established, while force and impedance control will remain future work that can build of the work from this project.

Due to the focus on software, basic software engineering principles can also be applied to improve maintainability and continued development. Reusable software development proves to also be an important but challenging aspect in the field of robotics (Brugali & Prassler, 2009), and it is also hoped that this project can demonstrate some advantages of noting its importance.

2. Literature Review

The PhD thesis from Dr Ma Zheng (Zheng, 2017) constitutes the basis of most of the work done on the current end-effector design. The thesis details the mechanical design, position control, impedance control, mid-ranging control, and some experiments conducted on the end-effector module, referred to as a mini manipulator. The figure below shows the preliminary design.

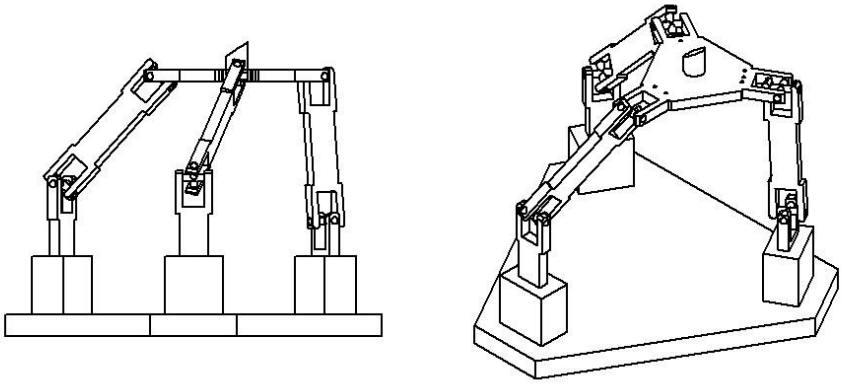


Figure 1: Preliminary Design of the End-Effector

The mechanical design affects the position control, shown mainly in the kinematics equations provided. These equations determine the possible positions of the end-effector and provide constraints that the software system will have to work with. An example of such equations is shown below, as the inverse kinematics equation.

$$offset_i = \sqrt{L^2 - x_i^2 - y_i^2}$$

$$z_i = \sqrt{L^2 - (x - x_i)^2 - (y - y_i)^2} - offset_i + z$$

where (x_i, y_i, z_i) denotes the coordinates of each limb, (x, y, z) is the task-space coordinate, *L* is the distance between universal joints of a limb.

These equations mean that it is easy to determine the position of each limb given knowledge of the exact coordinate of our task-space. Unfortunately, the same cannot be said for the

inverse kinematic equations, to determine the position of the task-space from the position of the limbs, as seen in the forwards kinematics equations (Zheng, 2017).

The following FYP student then proceeds to build the first prototype as seen in the image below (Wong, 2019). This prototype adapted most of its design from Dr Ma's first prototype, with minor modifications made.

Figure 2: First Mini End-Effector Prototype

The next FYP student focused on establishing an end-effector interface and display control module, making use of Raspberry Pi, analysing communication protocols, and designing housing for the Raspberry Pi (Thoe, 2020). Establishing communication, however, was not successful, and the author noted that there were compatibility issues involved.

At the same time, another student continues to develop an improved lead screw mechanism (Goh, 2020) used as the 3 limbs that would be used by the student after him. The mechanism increased stability and reduced the number of customized parts and is shown in the figure below. The Arduino Arduino Mega 2560 was chosen as the microcontroller board to control the motors. Basic Arduino code is also provided as an example, and this Arduino control proves to be the basis of this project.

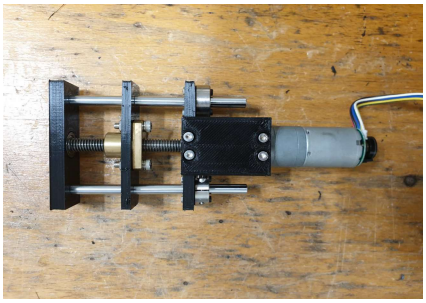
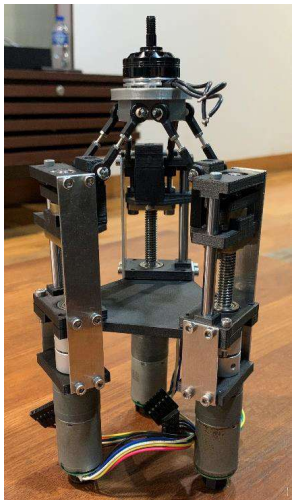


Figure 3: Lead Screw Mechanism Prototype

The mechanism is used by the next student, which will continue to be the prototype used for this project (Yang, 2021). This overall design was made to be more compact. An electronic schematic diagram was also provided, making use of a transformer, motor drivers, and an Arduino Mega 2560, all attached to the prototype with limbs made from prototype of the previous student. This diagram was used in this project in connecting the end-effector. Basic testing code was also provided.



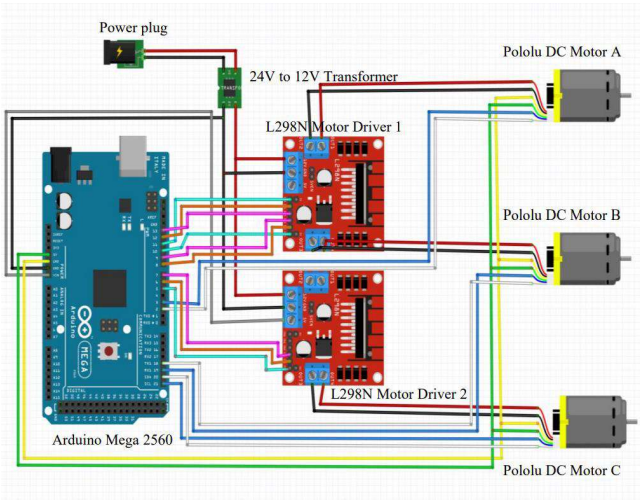


Figure 4(a): Second Mini End-Effector Prototype, Figure 4(b): Electrical Schematic

Wire Legend
Blue - EncA
White - EncB
Cyan - Enable
Orange - In1
Pink - In2

3. Methodology

From the project goals and timeline, there were 3 stages:

1. Initial investigation
2. Develop control methods
3. UI development

Due to the time constraints, and priorities set by Dr Joel, force control and impedance control was not worked on as control methods in favour of User Interface (UI) development. The focus was then pushed towards a robust position control with perhaps an intuitive Graphical User Interface (GUI).

The first step before conducting software design was to configure the electronic setup as shown in the earlier figure (Yang, 2021). The main components for this section are:

1. Polulu DC Motor (Polulu Corporation, 2021)
2. L298N Motor Driver (Handson Technology, 2018)
3. Arduino Mega 2560 (Arduino, 2020)

A breadboard was also purchased to be able to connect common nodes.

The next step was to be able to control the motors of the given setup. This made use of base testing code provided (Yang, 2021). Controlling the motors made use of the Arduino Encoder library (Stoffregen & Coon, 2021), which provides a way to use quadrature encoded signals to determine how much the motors have moved. These prove to be accurate, especially with the difference in strengths of the 3 motors used. Thus, using the encoders, it was possible to move the limbs up and down relative positions reliably, with small errors investigated later.

Since the end-effector needed to be controlled by a user, Arduino's Serial Monitor was used in order to pass these commands in (Arduino, 2021). This served as a Command Line Interface to be able to send commands to the motors directly. An example of such code is seen in the figure below.

```
// Check whether Serial is available
if (!Serial.available())
  return;

// Goes here if Serial is available
// Parse character and long integer to use as commands
char dir = Serial.read();
long t = Serial.parseInt();

switch (dir) {
case 'm':
  move_effector_to(t);
  break;
// More code here
}
```

Figure 5: Code Snippet of Arduino's Serial

Afterwards, code was then developed to control the motors. To improve code quality, repetitive code was abstracted out of the existing test code (Wadhwa, Rajapakse, Chia, & Soo, 2021). Working with C++ code meant that there was a need to use multiple source files, and hence header files to ensure maintainability, of which includes best practices (Robertson, Saether, Sharkey, & Rutherfurd, 2020). In terms of coding quality and standards, a style guide could indicate important concepts, making sure the code is optimized for the reader, and is consistent (Google, 2021). The project attempts to make use of the concepts when designing the code, including comments for non-trivial functions. Version control was also implemented with Git, and the code can be found on GitHub[1].

Initially, the design of the code made it such that only relative values for the encoder was processed. In the table below, each command causes a movement of the chosen value.

Table 1: Commands Implemented

Command	Function
m1000	Moves all actuators up until the corresponding encoders read 1000. Effectively moves effector up and down.
a1000	Moves actuator A
b1000	Moves actuator B
c1000	Moves actuator C
h1000	Moves the effector in a pseudo-hexagonal shape, by controlling individual/pairs of actuators.

[1] https://github.com/nus-wira/eg2605-urop

This design was later changed to use absolute values instead, to ensure that the program is aware of invalid values, after calibrating for the range of values. Furthermore, the program was made to automatically handle percentage values as well, shown in the table below.

Table 2: Parsing of Input

Input t	Parsed pos
$t < 0$	pos = 0
$0 \leq t \leq 100$	pos = t * maxEncVal / 100
$100 < t \leq maxEncVal$	pos = t
$t > maxEncVal$	pos = maxEncVal

The table above shows that the input *t*, referenced as well in the earlier Figure 5: Code Snippet of Arduino's Serial, would be parsed to have an absolute position value that would be used by the rest of the program. *maxEncVal* was determined empirically and is discussed in the next section. This is done after calibrating the effector, which involves moving the actuators to the lowest possible position and setting the absolute encoder values to 0. The function to do so is also included as an option for the user.

A user guide is also provided to detail the various commands available to the user. Such a document would allow any user a quick reference, no matter their skill level in understanding the code (Chafin, 1982).

4. Findings

As mentioned earlier, the encoder values were ultimately determined to be within a range of values after calibration and testing. From the calibrated value of 0 at the lowest point, the actuator was set to move up until it was not able to, and the encoder value of 50000 was printed. Thus, we have the inequality $0 \leq maxEncVal \leq 50000$. Conversion of invalid values was then done as such

$$maxEncVal = \min(maxEncVal, max(0, maxEncVal))$$

Table 3: Actual Encoder Values After Command 'm1000'

Command	Encoder	1	2	3	4	5	6	Ave	% Diff
m10000	A	10358	10342	10355	10400	10381	10404	10373	3.73
	B	10172	10181	10176	10188	10186	10177	10180	1.80
	C	10513	10513	10522	10529	10531	10536	10524	5.24

The table above then shows the actual encoder values after the command 'm10000' was sent to the program, from a starting position of $pos = 0$. While the program is set to stop at the exact value, which is $pos = 10000$ in this case, some deviation as shown above can be seen. The logic involved means that the motors are set to stop once encoder reading is close enough to the actual value.

Table 4: Time Taken for Commands Given

Encoder	Command	Time taken (s)			
		1	2	3	Ave
A	a50000	9.15	8.96	8.94	9.02
	a0	8.79	8.7	8.67	8.72
B	b50000	8.49	8.51	8.55	8.52
	b0	8.42	8.41	8.44	8.42
C	c50000	7.60	7.63	7.58	7.60
	c0	7.39	7.44	7.38	7.40

The table above then shows the time taken for each actuator to complete the command as shown, starting from the initial position of $pos = 0$ for the first command, after which the second command is issued. While the same model of motor is used on controlling all 3 actuators, it is seen that each motor takes a different amount of time to perform their given task.

A single movement test function was also added, showing how the actuators move together, under the function *test_full*(). This was a simple test, not accounting for any speed difference between the motors.

To represent the code, Unified Modeling Language (UML) diagrams were used, which provides an industry standard to visualize relationships in the code (Booch, Rumbaugh, & Jacobson, 2005). This project makes use of the Class Diagram and Sequence Diagram, which

are basic diagrams used in UML (Wadhwa, Rajapakse, Chia, & Soo, 2021). The header files closely modelled the relationship found in the class diagram below. The diagram shows that ui and logic portion of the code is found to be mostly self-sufficient, not requiring the knowledge of any of the other header files.

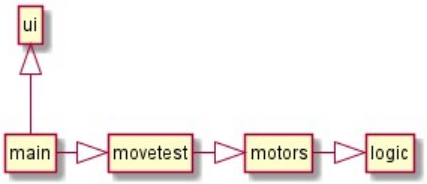


Figure 6: Class Diagram

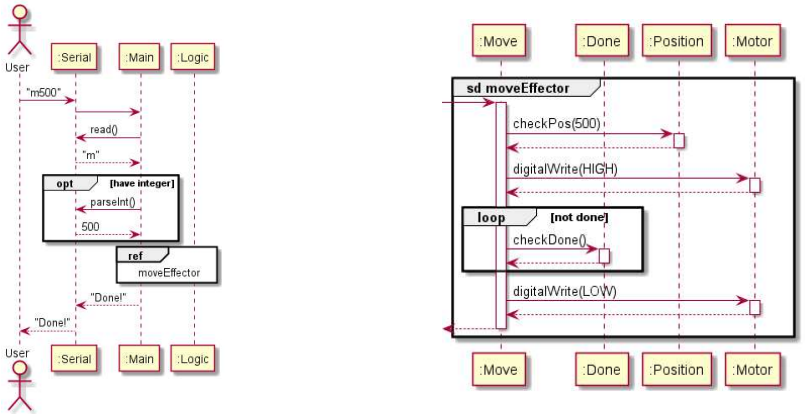


Figure 7: Sequence Diagram for Command "m500"

The diagrams above then shows the sequence diagram when a user inputs an example command "m500" to move the effector to $pos = 500$. It can be noted that *moveEffector* is a command that can move all 3 actuators at the same time, and can be used by various commands, serving as an abstracted function. For example, moving actuator A will also use *moveEffector*, which will proceed to only move actuator A, as actuators B and C are set to the position they were already at.

5. Discussion & Future Works

Occasionally, actuator B did not move even when the motor did, requiring slight adjustment of the position of the effector. This may have been due to friction between the motor and the screw mechanism not being great enough to ensure rotation. The figure below also shows parts that may get stuck when the actuators are moving up and down, colliding instead of moving past one another. It is noted that these issues tend to occur when the effector is on its side.

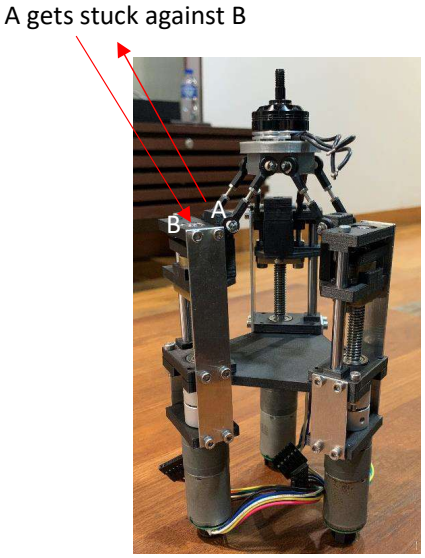


Figure 8: Parts Stuck on End-Effector

Currently, when calibrating the effector to determine *maxEncVal*, the actuators are required to move to the extreme end, and possibly physically push against the mechanical barriers, which may not be ideal. As can be seen in the user manual (Polulu Corporation, 2021), the higher resistance to motion will still draw current, effectively wasting energy with no movement. To improve on this, there could be the addition of a force sensor to detect resistance to motion, allowing the program to stop at the appropriate time.

The findings also showcase different results when moving the 3 actuators, which can be explained simply by the differences in the motors. The deviations from the chosen value of 10000 when issuing the command 'm10000' would be attributed by the momentum that the motors. While the code effectively sets the motors to stop right when the encoder values are

close enough, the motors will not immediately stop when the condition is reached. This means a small extra distance is moved. The 3 actuators also cover different distance which could depend on their speed when reaching the threshold, with actuator C covering the greatest distance. This leads directly into the next paragraph discussing the time taken for each actuator.

Each actuator takes a different amount of time when issued a similar command. This could be attributed to the motors having different amounts of wear and tear within their own components, allowing for each motor to reach different speeds. Table 4: Time Taken for Commands Given showed that actuator C took the least amount of time to travel its allocated distance, which does match the discussion in the previous paragraph. Since time taken to travel is inversely proportional to speed, with the similar distance covered, the speed of actuator C when reaching its threshold would be greater and would take a greater distance to bring to a stop. This assumes that friction for each motor would be similar, which may not necessarily be true.

More tests on the movement of the motors could also be done on top of the current implementation. These tests could involve the control of the motor speeds, plotting the x, y, and z trajectories (position vs time) at different control settings. More tests could be implemented if force and impedance control are also being worked on.

For the design of the code header files, it may be possible to implement unit tests for the source files that are separated from Arduino libraries (Elmore, 2012). This can prove to be extremely useful if the code were to grow, ensuring it continues to work after many changes.

The sequence diagram shows the basic abstraction used in the functions, including *moveEffector*, as discussed earlier. Other sequence diagrams or commands could simply build on these diagrams. Future work in this section would likely include more diagrams for more complicated functions if implemented such as those in discussed earlier in this section.

This project only showed a single sequence and class diagram, but future work could also include more diagrams, culminating in a developer guide, allowing for easier reference for

any future developers. The guide could include both mechanical and software aspects, where it is likely a developer would only be well-versed in one or the other.

6. Conclusion

This project aimed to continue work on an existing polishing end-effector, concentrating on the software aspect of position control. The paper discussed the previous work in the literature review, which mainly focused on mechanical aspects of the end-effector. It then discussed methodology behind the work done for this project, including the initial investigation, design of control elements, and UI development. These also included software engineering principles to increase maintainability of the project. Some discrepancies were found in testing of encoder values, attributed to difference in the motors. UML diagrams were also added to visualise the relationships of the software. Ultimately, future work can build off this project, adding better tests and future control, as well as a possible a developer guide as a reference.

References

Arduino. (2020). *Arduino Mega 2560 Rev3.* Retrieved from Arduino: https://store.arduino.cc/products/arduino-mega-2560-rev3

Arduino. (2021). Serial.available(). In *Language Reference.* Arduino.

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *Unified Modeling Language User Guide.* Addison-Wesley Professional.

Brugali, D., & Prassler, E. (2009, March). Software Engineering for Robotics. *IEEE Robotics & Automation Magazine*, pp. 9,15.

Chafin, R. L. (1982). User manuals: What does the user really need? *SIGDOC '82* (pp. 36-39). New York: Association for Computing Machinery.

Elmore, E. (2012, July 11). *Don't Run Unit Tests on the Arduino Device or Emulator.* Retrieved from StackOverflow: https://stackoverflow.com/a/11437456

Goh, Q. E. (2020). *Design completion of a light machining end-effector.* National University of Singapore: Department of Mechanical Engineering.

Google. (2021). *Google C++ Style Guide.* Google.

Handson Technology. (2018). *L298N Dual H-Bridge Motor Driver.*

Li, J., Guan, Y., Chen, H., Wang, B., Zhang, T., Liu, X., . . . Zhang, H. (2020). A High-Bandwidth End-Effector With Active Force Control for Robotic Polishing. *IEEE Access*, 169122-169135.

Polulu Corporation. (2021). *25D Metal Gearmotors.* Las Vegas: Polulu Corporation.

Robertson, C., Saether, L., Sharkey, K., & Rutherfurd, A. (2020). *Header files (C++).* Microsoft.

Stoffregen, P. J., & Coon, R. C. (2021). *Encoder Library.* Oregon: PJRC.

Thoe, W. (2020). *End effector interface and display control module.* National University of Singapore: Department of Mechanical Engineering.

Wadhwa, B., Rajapakse, D. C., Chia, H., & Soo, Y. (2021). *Software Engineering for Self-Directed Learners.* National University of Singapore: School of Computing.

Wong, K. T. (2019). *Development of a Compact Robot End-effector for Light Machining.* National University of Singapore: Department of Mechanical Engineering.

Xu, P., Cheung, C.-F., Li, B., Ho, L.-T., & Zhang, J.-F. (2017, April). Kinematics analysis of a hybrid manipulator for computer controlled ultra-precision freeform polishing. *Robotics and Computer-Integrated Manufacturing*, 44-56.

Yang, Z. (2021). *Design Completion and Testing of a Light Machining End-effector.* National University of Singapore: Department of Mechanical Engineering.

Zheng, M. (2017). *Control Framework for a Macro/Mini Manipulator.* National University of Singapore: Department of Mechanical Engineering.