KING'S College LONDON

University of London

Design and implementation of an adaptive controller for a robotic worm

Final Project Report

(BSc Computer Science with Intelligent Systems)

Author: Tomas Vitek

Supervisor: Dr Hongbin Liu

Student ID: 1156798

22 April 2014

Abstract

Endoscopy is an important medical procedure for health prevention. However the current endoscopes bring quite substantial discomfort to patients, because they require to be forced against the examined tissue.

An increasing research is being done in replicating peristaltic locomotion found in natural worms. A robotic worm using peristalsis can make the procedure of endoscopy shorter and less discomforting.

This report aims to design and develop an adaptive controller for such robotic worm and utilise this controller to experimentally determine the faster movement pattern to propel a segment-based robotic worm.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Tomas Vitek

22 April 2014

Acknowledgements

I would like to thank my supervisor, Dr Hongbin Liu, who has provided me with invaluable help and support throughout the project. I would also like to thank Thomas Manwell for his feedback.

To my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project Motivation

Endoscopy is a medical procedure of looking inside a human body with a thin long flexible instrument (called an endoscope), usually performed by inserting the endoscope in the colon or the gullet. Endoscopy is a very important part of health prevention as well as a useful tool, which assists surgeons during operations. The existing endoscopes bring quite substantial discomfort to patients, because to move the endoscope forward it requires an operator to "twist" and "force" it against the tissue of the organ in which the endoscope is inserted. This might result in irritation and possibly even perforation of the surface of the examined organ. This also means that endoscopy is time consuming and requires an experienced operator.

By designing a device, which can perform endoscopy automatically and autonomously, this procedure can be made less discomforting for the patient, it can be shortened and it can also remove the requirement of an experienced operator for such procedure.

1.2 Aims and Objectives

The main objective of this project is to contribute to development of an autonomous worm-like robot, which would provide much needed improvements in endoscopy procedures. This project aims to create a control circuit and software to enable easy testing of such worm-like robots with peristaltic locomotion.

Another objective of this project is to use the created control circuit and software to experimentally determine what is the fastest control pattern to propel a segment based robotic worm.

This project will build on previous research. Particularly the control algorithms will be tested on a robotic worm with peristaltic locomotion developed by Thomas Manwell [13].

Chapter 2

Background

This chapter of the report contains a summary and analysis of the background for this project. It includes a brief introduction into biology of peristaltic locomotion, which was an inspiration for the actuation of the proposed robots, as well as an overview of the up to date research.

2.1 Biology of Peristaltic Locomotion

Peristalsis is a type of movement found in nature [7], where a muscle contraction and relaxation propagates through a long thin muscular tube-shaped body, which causes the body or its contents to move forward. Peristalsis is used by earthworms for their locomotion. Figure 2.1 shows sequences of an earthworm going through peristaltic locomotion.

2.2 Research of Peristalsis in Robotics

The flexibility and the tube-like shape enable earthworms to move through very constricted and curved spaces [7]. This means that a robot replicating peristaltic locomotion could provide a good way to automate and improve on current endoscopes [5].

Because such robot uses same type of movement as the examined organs (colon, gullet, etc.) and because it does not require any external forces, the

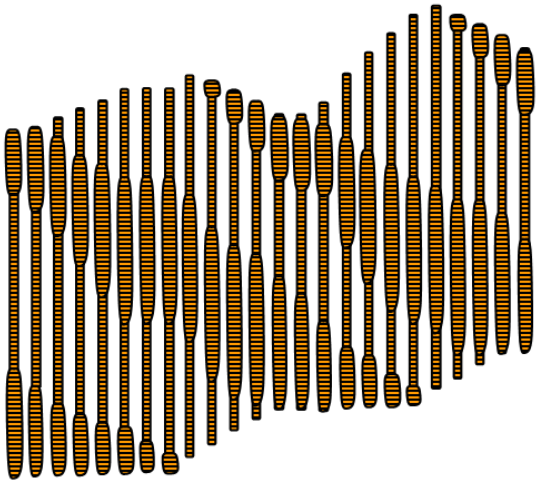


Figure 2.1: Sequences of an earthworm going through locomotion [12].

procedure can be performed less painfully for the patient and also does not have to require an experienced operator.

An increasing number of research and literature exist on replication of peristaltic locomotion. Dario [5] reviews these different locomotion techniques for robotic endoscopy and states that they could revolutionise the current endoscopy techniques.

2.3 Actuation Methods

Many different authors present different types of actuation to achieve peristaltic locomotion.

For example, Menciassi [14] shows several steering systems made with Shape Memory Alloy, which is made to change its shape when applying energy (usually heat or electricity) to it, but remains static when no energy is being applied. Figure 2.2 shows a typical segment of a worm-like robot with Shape Memory Alloy and clampers and extensor (which contract and relax the alloy).

Another locomotion mechanism, proposed by Kim [11], propels the worm by elliptic motion of multi legs. Figure 2.3 shows such design.

Manwell [13] proposes a worm-like robot made from braided mesh with

Figure 2.2: A still from a video of a worm robot made from Shape Memory Alloy at MIT [4].

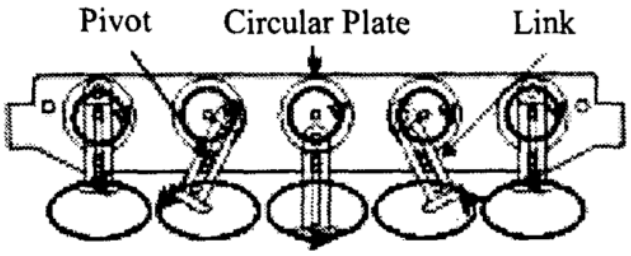


Figure 2.3: Principle of the Ellipsoidal Motion of Legs as proposed by Kim [11].

folds actuated by threads wound on a DC motor.

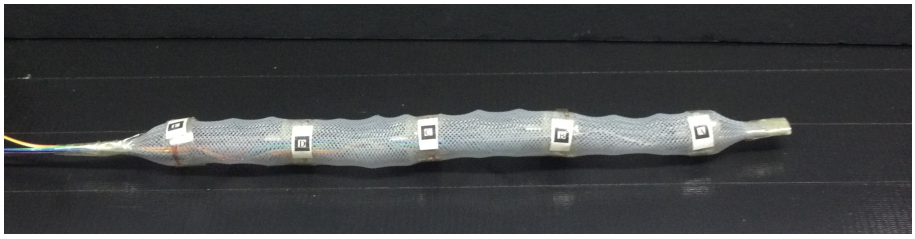


Figure 2.4: Photograph of a fully extended mesh worm robot made by Manwell [13].

2.4 Research in Movement Patterns

When controlling the robotic worm with peristaltic locomotion a movement pattern is used. This pattern defines the peristaltic wave (how the contracting and relaxing of individual segments moves through the worm).

Although much research has addressed the hardware design and fabrication of a robot driven by peristalsis, most of the up to date research focuses on designing and constructing the worm, but usually uses only one movement pattern, which results in inability to compare different movement patterns as they are all implemented for different robotic worms and as such the comparison would not be objective.

This means that the question of the optimal control pattern still remains unaddressed. This project aims to address such question.

2.5 Controlled Robotic Worm

Since this project does not aim to design and construct a new robotic body, but rather a control system for one, it will be developed based on an already constructed worm-like robot made from braided mesh with folds created by Manwell [13].

Choosing to use already developed robot body does allow me to focus on the important part of this project — to determine the optimal control patterns.

This robotic worm consists of variable number of segments. Each segment is equipped with one 10mm DC Gear motor by Precision Microdrives (model 210-002) (for the datasheet please see appendix F). A number of threads are tied from one end of each segment to its motor on the other end of the segment. When the motor rotates, the threads wind around it causing the segment to contract. When the motor spins in the opposite direction, the folded braided mesh of the segment acts as a spring and extends the segment back to its initial position. Figure 2.5 shows a segment fully extended and contracted.

Each of the segments provides negative and positive wires for driving its motor. This project will create a controller, which will interface with these

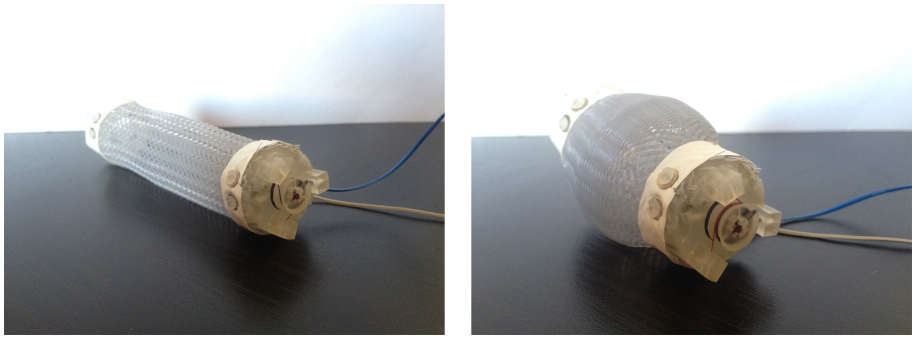


Figure 2.5: Fully extended worm segment (on left) and fully contracted worm segment (on right).

motors to control the worm.

Chapter 3

Requirements

In this chapter of the report, I have listed all requirements according to which the system should be developed. If successful this project should result in creating a controller for worm-like robot as well as provide a platform on which future developments can continue.

The requirements are presented in no particular order.

3.1 User Requirements

User requirements define all actions the user must be able to perform when using the system; the user must be able to:

1. Control individual segments of the connected worm robot.
2. Automatically load and run different movement patterns on the connected robot.
3. Start and stop the worm whenever it is required.
4. Collect data about the state of the worm to be able to compare different movement patterns.

3.2 Functional Requirements

Functional requirements define a list of functionalities the system has to implement; the system must:

1. Be able to connect to a segment based worm-like robot with DC motors.
2. Provide the user with a GUI to interact with the system.
3. Enable to load a movement pattern configuration and automate the execution of it.
4. Provide the user with a way of controlling individual segments.
5. Enable the user to start and stop the worm on request.
6. Produce a data visualisation of the current state of the worm to provide the user with a mean to compare different movement patterns.

3.3 Non-Functional Requirements

Non-functional requirements are set to ensure overall good quality of the system:

1. The system should be reliable. It should be able to replicate the same results with the same movement patterns.
2. The system should be responsive. When the user makes an action, the system should either execute such action or provide the user with feedback to explain why it is not possible.
3. The size of the system should be minimised to enable portability of the system.
4. The user interface of the system should be easy and intuitive to use.
5. The software part of the system should be able to run on most common operating systems.

6. The design and implementation of the system must meet the highest professional and ethical standards, which include the ones addressed by the British Computer Society Code of Conduct.[1]

[1] http://www.bcs.org/category/6030

Chapter 4

Design and Specification

Following the stated requirements, this chapter discusses the design decisions made in the process of developing the system and it details what each part of the system will do. Each design choice is either motivated by the overall goal of the system or addresses one of the set requirements.

4.1 Project Structure

In order to provide the user with control over the worm robot, the system will need to interface with both the hardware of worm and the user. This means that the project will be consisting of two components:

- A control circuit will be designed and build to interface with the robot and its motors, which power each segment. This control circuit will be directly controlling the speed and the direction of a motor for each segment of the worm.
- A software control application will be developed, which will provide an interface to the user. Based on their input and its control algorithm, it will send commands to the control circuit, which will based on these commands control the worm.

Figure 4.1 shows the component of the system.

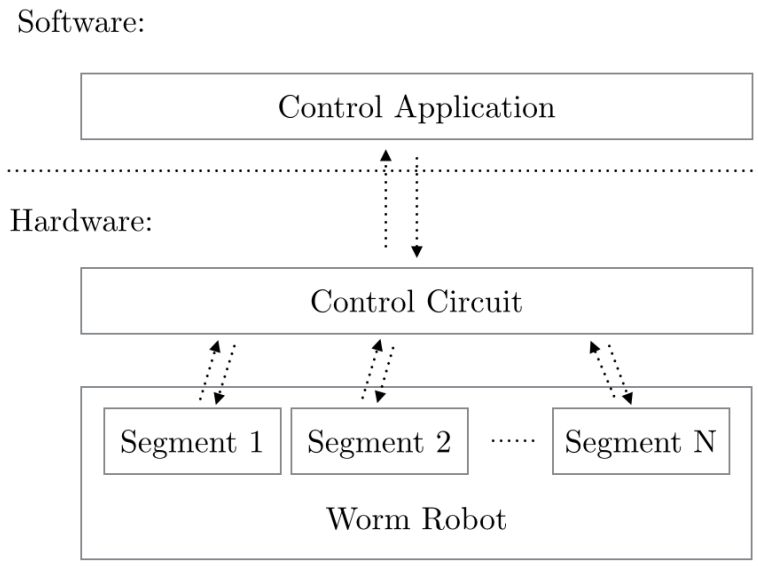


Figure 4.1: Components of the system.

While the majority of the control algorithm of the system could be implemented in hardware, I have decided to implement as much of the system as possible in software. This is because software development, unlike hardware development, enables a rapid pace of prototyping and going from an idea to testing that idea.

Another advantage of this approach is that the development is much cheaper, as it does not require as many hardware parts. However such approach also has one disadvantage — the speed of the system could be impaired, as converting the analog signal to digital and sending it to the computer to process and converting it back again requires time. It is one of the challenges of this project to design it such that this does not occur.

The following sections of this chapter describe the design decisions for both components of the system.

4.2 Design of Control Circuit

Given the requirements for the hardware part of the control system, the most suitable solution is a single-board computer or micro-controller. These small devices provide a full featured programming language to enable a fast development of a system. They enable easy prototyping with a variety of available expansion boards and are available for a reasonable price.

4.2.1 Micro-controller

Amongst the most known such devices are Raspberry Pi [15] and Arduino [2]. Raspberry Pi is a microcomputer, which can run a full featured operating system, such as Linux.

However for the purpose of this project Arduino board will suffice, as it is capable of driving the motors of the individual segments and can also easily communicate with a PC, wirelessly or over a connected cable.

Arduino provides multiple models of its micro-controllers, with varying amount of input and output pins. Because the control circuit should be capable of controlling a robotic worm with a varying number of segments, Arduino Mega [1] board has been chosen, as it provides 54 digital pins, which can be used for both input and output, which means that the control circuit can be extended later to accommodate for larger number of controlled segments.

4.2.2 Motor Drivers

The Arduino does not offer an option to drive a DC motor directly from the board. There are expansion boards available for Arduino, which provide this functionality [17], however these boards are sized 3 x 6 cm, and since this project will require driving at least 4 motors, multiple expansion boards would be required, which would make the overall size of the control circuit considerably larger.

After studying a schematic diagram for the Arduino Motor Shield (available at [17]), I have found out that the expansion board uses a L298 H-bridge motor

driver (appendix G). In the effort to keep the control circuit as small as possible, it has been decided to use these L298 motor drivers.

Each L298 driver (appendix G) can drive up to 2 motors, however they share common voltage supply, which means that when both of the motors would be turned on, the common voltage supply would be able to provide each of the two motors with just a half of the available voltage. Increasing the voltage supply in such case is not desirable, because when just one motor is running and the other is idle, all available voltage can be supplied to the running motor. Therefore increasing the total available voltage could damage the motors.

Solution to this problem is to use one L298 motor driver for one motor. This ensures that each motor can draw as much voltage as it requires, but never more that it could damage it.

4.3 System Controller

There are two fundamental types of system controllers:

1. **Open Loop Controller**

   does not use any system feedback. This means that the output of the controller is computed only based on the internal model of the system and its environment. While such system can be easier to design and implement, it is not able to correct errors if such occurs. This means that an open loop controller is not suitable for complicated or unpredictable environments.

2. **Closed Loop Controller**

   does use system feedback. The system computes its output not only based on its internal model, but also on the measured feedback. The system uses this feedback to check if it is in the desired state and if not, it corrects the error to make sure the system reaches the desired state. Such controller requires an ability to measure or read some information about the system, usually using a set of sensors.

Since the environment of the robotic worm is quite complicated and possibly unpredictable, I have decided to design a closed loop controller, which will utilise system feedback to make sure it performs correctly and with minimal error.

Because the robotic worm [13] does not have any sensors, the only available feedback is a current feedback from its motors. While it would be possible to modify the body of the worm robot to include additional sensors for collecting state feedback (such as potentiometers or flexible resistance sensors), doing so would increase the overall weight and complexity of the system, resulting in slowing the worm down and increasing the probability of an error. As a result, I have opted to use only the current feedback already available to me.

The L298 motor driver provides a control pin, which can provide an analog signal based on how much current does the driven motor draw. The control circuit will utilise this functionality to provide the current feedback to the worm control application.

The design of the closed loop controller is detailed below in the Control Application section of this chapter.

4.4 Communication between Control Circuit and Control Application

Arduino board provides a Serial interface [16] for communication with a connected computer over USB cable. Since this Serial communication interface is standardised and it is supported by most high level programming languages either natively or as an external library, Serial interface will be used as a mean to transfer data between the control circuit and the control application.

Adriano will function as an intermediary between the hardware of the control circuit and the control application. Since most of the control algorithm will be implemented in the worm control application running on a computer, only the following functionalities will be implemented for the Arduino board:

- Ability to receive and send commands to the control application over the

Serial port.

- Ability to turn individual motors on and off, change their speed and direction based on received commands.
- Ability to read the current feedback from the motor drivers and send it to the control application over Serial port.

Figure 4.2 shows a diagram of communication between the control application and the control circuit of the system.

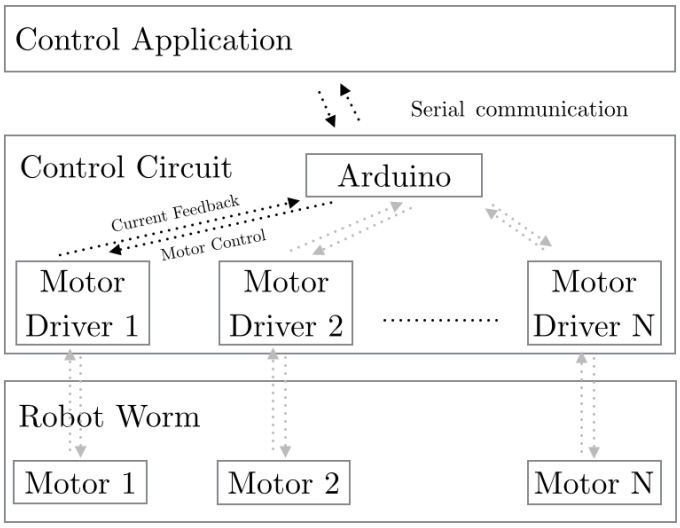


Figure 4.2: Communication between the compoments of the system and the connected robot.

Communication Protocol

To make communication between the control application and the Arduino easier, a simple communication protocol has been designed. Both the control application and Arduino will be able to send its counterpart a message over Serial port to request an action or deliver some information.

Serial protocol can signal an application when it is receiving a message over the port. Such application can then read one or more characters from

this message. The system will utilise this and it will use a simple message format to interchange commands from and to Arduino.

Each message will have the following format:

```
[command:arguments]
```

Character "[" represents the start of a message and character "]" represents its end. Upon receiving the end character, the recipient of the message will parse its command and arguments, and will execute an appropriate action according to the command.

An alternative command with no arguments can be send as well:

```
[command]
```

4.5 Worm Control Application

Arduino provides a C-based programming language, which would enable the control algorithm to run directly on an Arduino board, however this would not provide any feedback to the user, as Arduino does not have any built in display.

To address this problem, a graphical user interface (GUI) application will be developed for personal computers. This application will collect input from the user and based on its control algorithm, it will send commands to the control circuit, which will directly control the robotic worm.

4.5.1 Application Platform

Java has been chosen as a platform for developing the worm control application. Reasons for selecting this programming platform follow:

- Java is multi-platform. An application written in Java can run on any common PC platform (Windows, OSX or Unix).
- Java provides very good tools for rapid development and fast debugging.

- Using the Swing library, it is easy to create a GUI interface for an app written in Java, and it does not require any additional dependencies.

Other programming platforms have been considered as well, but they do not provide any additional advantages over Java for this project.

4.5.2 Software Development Process

Because software of the project will be closely coupled with its hardware and cannot be developed separately, I will mostly follow the spiral model for software development of this project. Firstly implementing a hardware part of a feature and then developing its software counter part.

Also I will be testing the functionality of both the control application and the control circuit during the development of the system.

4.5.3 Closed Loop Controller

As I have outlined above in the System Controller section of this chapter, I have decided to design and implement a closed loop controller with current feedback from the driven motors.

The controller will process and store the current feedback signal from Arduino and it will utilise this information to control the motors of the worm.

There will be two different layers of the controller, which will both control the worm.

The aim of the first layer of the controller will be to based on the current feedback determine the state of the controlled segment, to decide if the segment has reached its desired position and needs to be stopped or if it has not yet reach the position and needs to remain moving.

The second layer will then use the obtained current feeback to determine the speed of the driven motor. These two layers could be integrated into one control loop, however to provide the user with more control over the worm, I have decided to implement the speed control separately and enable the user to adjust its parameters.

Since each segment of the worm needs to be controlled separately, there will be one instance of the controller for each segment.

Next I will describe each layer of the controller in detail.

Segement State Recognition

To reliably determine the state of a controlled segment a simple pattern recognition will be used.

To be able to find a viable pattern to look for, I have performed tests during which I have measured the amount of drawn current when a segment is manually closed and opened.

In these tests a segment has been manually contracted ("closed") until it reached its fully closed state and then it has been immediately relaxed ("opened"), this was repeated two times per a test. The current feedback has been recorded during these tests. The tests were repeated three times, always yielding similar results.

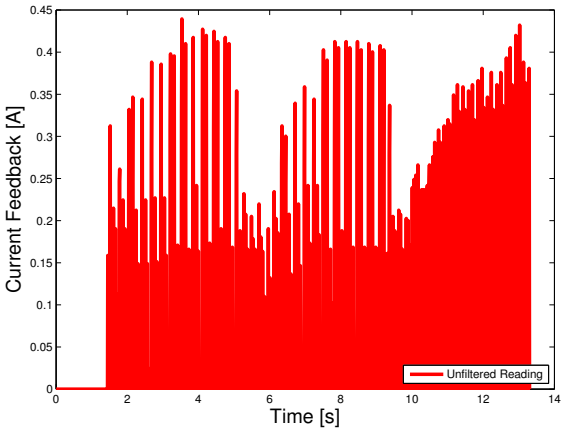
Figure 4.3 shows plots of the measured current from the tests.

As figure 4.3 shows there is quite a lot of noise in the measured signal. However an overall pattern can be seen.

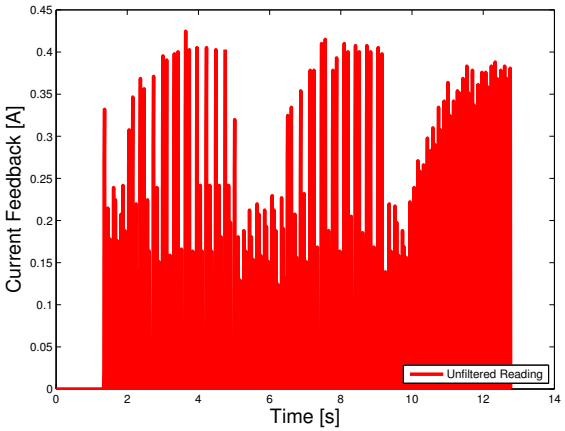
A DC motor draws current in direct proportion to its load. If the load increases, the motor draws more current to be able to accommodate for this increase and vice versa, when the load decreases the motor draws less current. The measured current in this test has confirmed this.

As a segment is closing its load is increasing, since it has to provide more force than the mesh pushing against it (the mesh effectively acts as a spring), therefore the motor draws more current. When the segment is fully closed, the motor cannot rotate any further (as the segment can't close anymore), which causes the motor to stall. This is reflected in the measured current as local maxima..

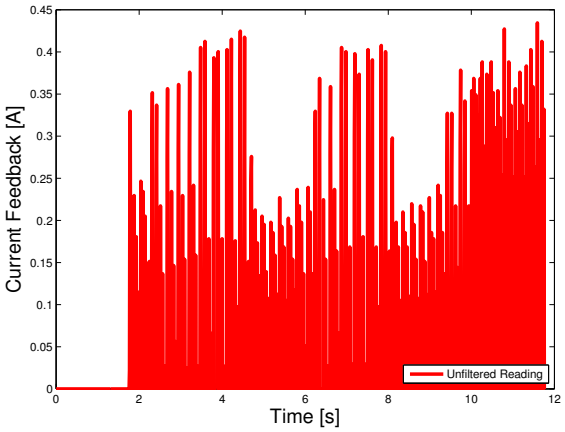
Similarly when the segment is opening, its load is decreasing, resulting in decrease of drawn current. When the segment is fully opened, the load on the motor is the smallest, as the mesh is fully extended and therefore is not



(a)



(b)



(c)

Figure 4.3: Plots of measured current drawn during tests when manually closing and opening segment.

pushing against the motor. However if the segment would be to "overshoot" (miss the point where it is fully opened), its motor will continue to rotate in the same direction, which will cause the wound thread to fully unwind and start winding in the other direction, which will cause the segment to start closing again. This is reflected in the measured current as local minima.

To determine the state of a segment, the controller will use this pattern — it will look for peaks (local maxima and minima) in the current feedback signal, to decide when if the segment is fully closed or opened.

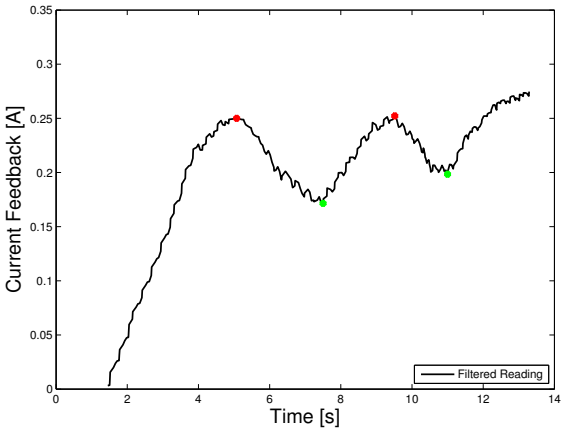
Noise Filtering

To be able to recognise patterns in any signal, this signal needs to be without noise. As it is evident from the figure 4.3, the measured current feedback contains quite a lot of noise.

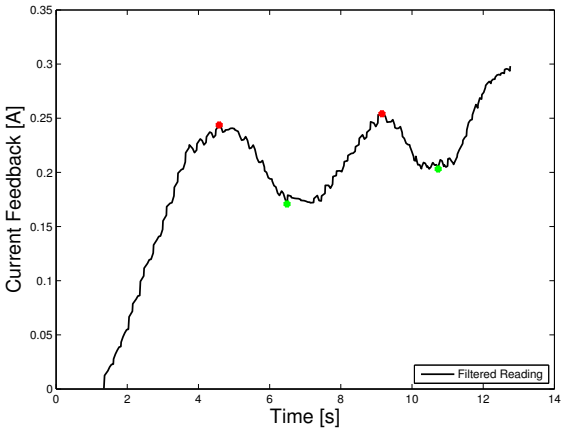
A filtering system needs to be implemented to remove the noise from the measured signal to ensure that the pattern recognition can be as reliable as possible.

However if the filter will be too slow, it could cause a time delay in the signal, which would affect the reaction speed of the pattern recognition. The filter needs to react to changes in the signal fast enough to avoid this problem.

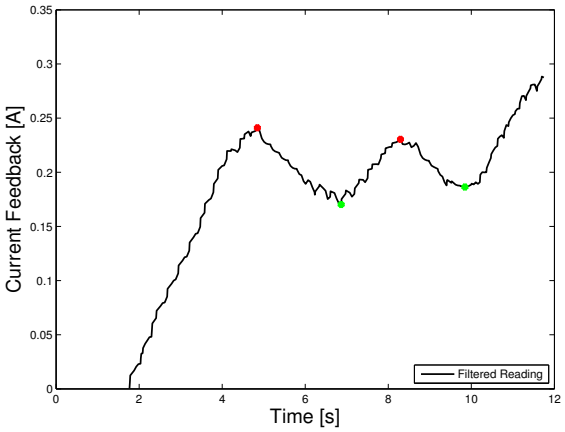
I have tested several different filtering techniques, but a simple smoothing filter (running average) produces good enough results, which do not delay the signal significantly. The current feedback of the system will therefore implement smoothing as a noise filter. Figure 4.4 shows the measured current from figure 4.3 filtered by running average of last 20 samples, the plot includes found local maxima and minima (plotted by red and green dots).



(a)



(b)



(c)

Figure 4.4: Plots of filtered current drawn during tests when manually closing and opening segment.

Speed Control

When the Arduino board is controlling a DC motor using the L298 motor driver, it can set an arbitrary speed for the motor from 0 to 100 (in percent of the available voltage). This means that the control algorithm can include a variable speed.

One approach would be to allow the user to set a speed and then run all motors with this defined speed. While this would result in the worm going slower (or faster), it is the same as decreasing (or increasing) the available voltage for the system and therefore it does not provide any additional value, since the user can directly do that.

However an adaptive speed controller can be implemented. It would decide the speed of the motor based on the current state of the system. After consideration, I have decided to implement a variable proportional-integral-derivative (PID) speed control.

Åström [3] describes a PID controller as a three-term controller, which calculates the error by substracting the measured feedback from a desired value (which is often called a set point), the controller then attemps to minimise this error by producing a sum of the three terms:

1. **Proportional P term**

   The P terms reflects the current error.

2. **Integral I term**

   The I term reflects the accumulation of the past errors.

3. **Derivative D term**

   The D term reflects an estimate of a future error.

$$u(t) = K(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}) \tag{4.1}$$

The PID controller is more formally described by equation 4.1 and diagram 4.5, where y_m is a measured feedback variable, e is the control error (calculated

as $e = y_{setpoint} - y_m$). The PID controller has three parameters — proportional gain K, integral time T_i and derivative time T_d.

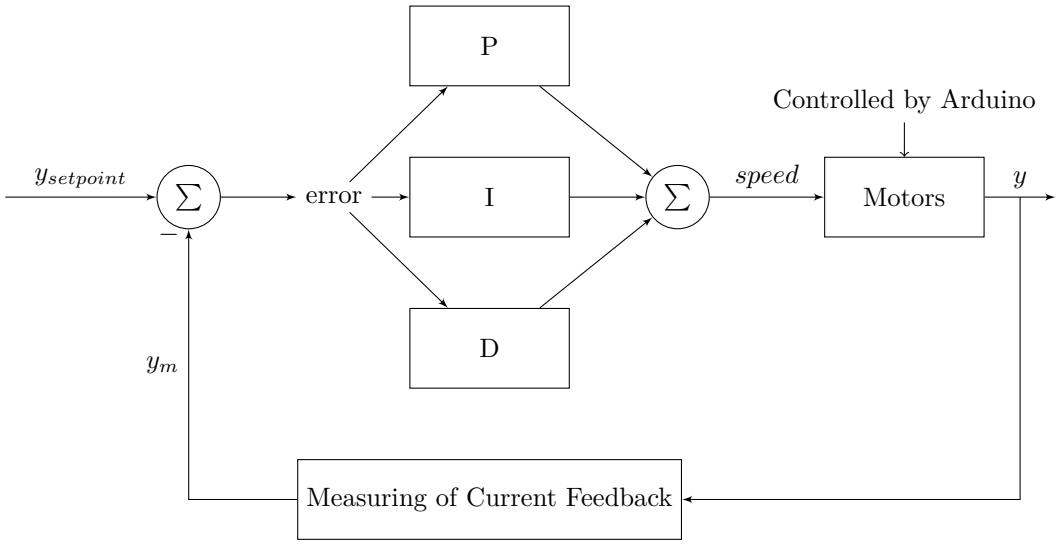


Figure 4.5: PID Control Diagram

To enable the user to define how fast or slow should segments close and open, all parameters of the PID controller will be adjustable, including all gains as well as the input and the setpoint in form of equations, which will be evaluated for every single computation of the PID controller.

To compensate for the fact that when a segment is closing, the load of its motor is increasing and when it is opening, the load is decreasing, the PID speed control for opening segment will be direct (the closer the input of the PID gets to its setpoint, the slower the speed will be) and the PID speed control for closing segment will be reversed (the closer the input gets to the setpoint, faster the speed will be).

By implementing an adjustable PID controller, any desired change of speed in time can be achieved.

For example, the user wants to slow the segment down as it is approaching its fully open state, they can use the fact that the pattern recognition for current feedback is searching for local minima and maxima (where current feedback gradient is equal to 1). If they set the input equation for the PID to current feedback gradient (current value / previous value) and the setpoint to

1, the resulting speed control will slow down as the segment is being opened (or speed up as the segment is closed, because the PID control for closing is reversed).

4.5.4 Segment Control

The user should be able to control each segment individually. A segment can be controlled either manually, where the user controls the motor of a segment directly, or the user can use automatic control to tell the segment to open or close.

Manual Control

This type of control is quite simple to implement, as it does not require any system feedback.

Using the communication protocol as outlined above, the control application will send a command to Arduino with information about how long and in what direction each motor should run. The Arduino board then after receiving this command will change the speed and direction of its motors appropriately. After this action is executed, the Arduino will send a command back to the control application, informing it that the action has been completed.

Automatic Control

While being able to manually control a motor of a segment can be helpful at times, in most cases the user will want to either fully close or fully open a segment.

To achieve this, the above described closed loop controller will be used.

The control algorithm will send a command to Arduino to turn the motor of the controlled segment on. The Arduino board will do that and it will start sending current feedback to the control application (and it will keep doing so until it has been given command to stop). The application will then based on this feedback determine whether the segment has reached its goal state (fully opened or closed), if not it will compute the speed for the motor of the segment

using the PID controller and send the speed to the Arduino, which will adjust the speed of the motor. This will continue until the segment has been fully closed or opened.

4.5.5 Pattern Control

However the main focus of this project is to create an adaptive controller to automate the movement of a robotic worm. To do so, I will design a controller based on predefined movement patterns.

Firstly we need to define what is a movement pattern in the context of the robotic worm. As described in the Background chapter of this report, in general a worm movement pattern define the peristaltic wave — how the contracting and relaxing of individual segments moves through the worm.

In case of the robotic worm, each segment can be either opened or closed (based on its current state) or it can be idle, therefore a movement pattern is a sequence of actions defined for all segments.

For example, if we would want a two-segment robotic worm to open the first segment and then as the first segment is closing to open the second segment, we would define the following movement pattern:

```
Pattern1 = {{Segment1: open, Segment2: idle}, {
    Segment1: close, Segment2: open}}
```

Figure 4.6: An illustration of a movement pattern configuration.

Movement patterns can be looped, resulting in a continuous movement. However the above pattern cannot be looped, as the second segment never closes, therefore to make sure that such pattern would work when looped, we need to ensure that the state of the worm at the end of the pattern is the same as at the beginning. To do so, we can modify the pattern in the following way:

```
Pattern2 = {{Segment1: open, Segment2: close}, {
    Segment1: close, Segment2: open}}
```

Figure 4.7: An illustration of another movement pattern configuration.

Now the pattern will loop correctly and it will generate a worm movement, when both segments 1 and 2 alternate in opening and closing. However this pattern requires that the first segment is closed and the second segment is opened before we start such pattern. To remove this requirement, if the segment is already opened (or closed) and is supposed to be opened (or closed), it will remain idle otherwise.

To make a configuration of a pattern more concise, the following pattern configuration syntax has been created:

```
O C
C O
```

Figure 4.8: An example of the final syntax of movement pattern configuration.

Character "O" means open, character "C" means close, character "X" means idle. Each line defines actions for each segment. The system then executes actions column by column. In case of this pattern, it will start by executing column 1, with "O" for segment 1 and "C" for segment 2. After segment 1 is opened and segment 2 closed, it will move on to the next column, with "C" for segment 1 and "O" for segment 2 etc.

The user will be able to save a movement pattern to a text file using the above-described syntax and load it using the control application. The system will parse this file and then it will enable the user to execute this pattern on the connected worm. It will function the same way as the automatic control of a single segment, but instead of controlling just one segment in such way, multiple segments will be controlled and more actions will be executed in sequence (based on the loaded pattern).

In same cases the worm segments do not have to be of the same length, which means that opening one segment might take longer than opening another. Also because the mesh is pushing against the motor of the segment when it is closing, it will take longer to close a segment than to open it.

To address this, the worm will not start executing another set of actions, unless the current set of actions has finished. So for example in the above

defined pattern 2, even if the segment 2 has already finished closing, it won't start opening until the segment 1 has also finished its current action (opening). This will prevent segments from "running ahead" of the pattern.

However it can happen that for some reason the segment won't be able to finish its current action (for example because of an obstacle), in such case the whole worm would stop and wait until this segment has finished. To prevent this, if more than one segment is executing an action at one time, when all segments are finished executing but the last one, a timer will be set to a certain time, after this timer has elapsed, even if the last segment has not finished, its current action will be terminated and the worm will move on to the next set of actions. This will make sure that the system will not become immobile.

4.5.6 System State Visualisation

To provide the user with information about the state of the worm, several graphs will be made available to the user. Three graphs will be implemented in total:

- Current feedback graph. The system will be able to display live current feedback data as it is being received from the Arduino board.
- Visualisation of the current state of the worm. A graph showing which segments are closed and which are opened.
- Movement pattern graph. The system will be able to generate a graph describing the user loaded movement pattern.

Chapter 5

Implementation and Testing

Having gone through the design of the system, this chapter will describe how is each part of the system is implemented and what tools or techniques have been used to developed it.

5.1 Hardware

5.1.1 Prototype of Control Circuit

To get myself familiar with the Arduino, before developing the full-featured control circuit, I have developed a circuit prototype with an Arduino Uno board. Since my intention was to learn how to drive a DC motor using Arduino, for the purpose of the prototype I have not used the L298 motor driver, but rather a cheaper alternative L293, which offers the same functionality, except it does not allow for current feedback. Figure 5.3 shows the developed prototype.

5.1.2 Control Circuit

Because the controller created for this project will be tested using a robotic worm with four segments, the final control circuit needed to be able to drive 4

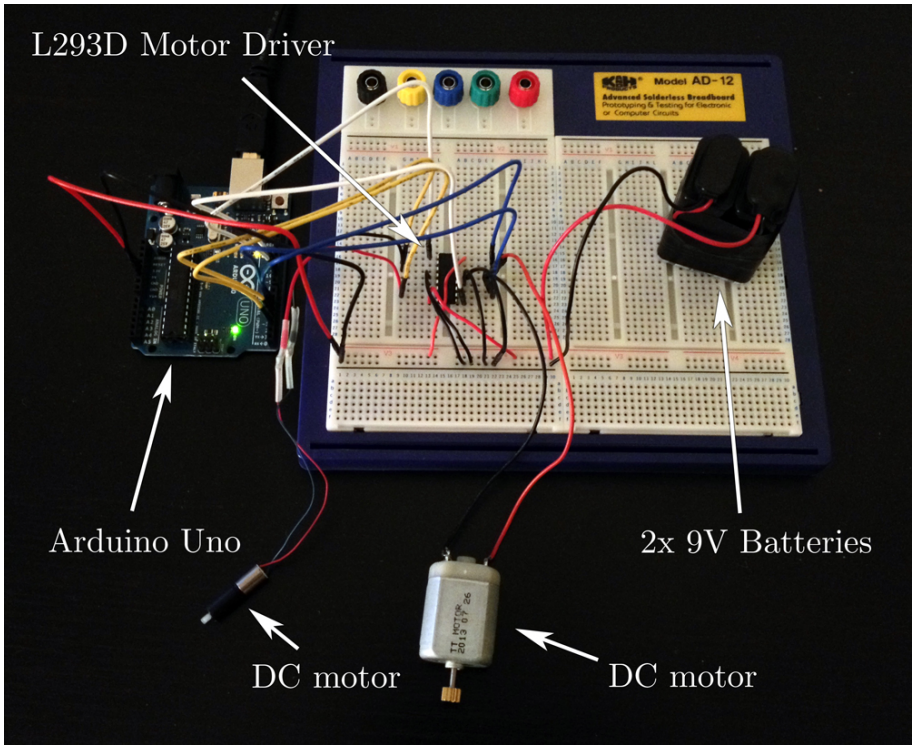


Figure 5.1: Prototype of control circuit with 2 DC motors and L293 motor driver.

motors.

To implement this, the L298 motor driver, which works in the same way, has replaced the L293 motor driver from the prototype, however it provides an additional pin for reading the current feedback. In the next phase the prototype has been extended on a larger breadboard.

To drive each of the motors, four wires are required - three to control the motor via the motor driver (one to set the speed and two to set the direction) and one wire to read the current feedback from the motor driver. This means that to drive 4 segments, each with one motor, 16 input and output pins will be required. To accommodate for this, the Arduino Uno from the prototype, has been replaced by larger model, Arduino Mega 2560.

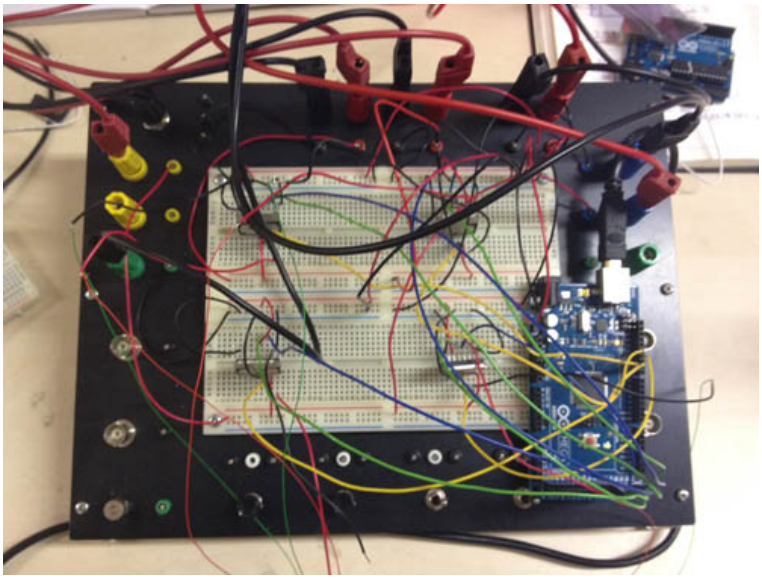


Figure 5.2: A final version of the control circuit controlling four motors.

Power Supply

Each of the motor drivers requires two different power supplies, one for its logic, which is supposed to be 5V and one for the driven motors. As it has been discussed in the design chapter of this report, each motor driver will drive only one motor, to avoid power supply sharing between two motors (which would cause decrease in efficiency.

In the developed prototype the logic supply for the motor driver was provided by the Arduino board, which offers a 5V power supply output pin. However when this 5V was shared between the four motor drivers in the extended control circuit, the Arduino board was not able to provide enough voltage. To address this issue an external fixed 5V power supply has been used. This 5V is then shared between all motor drivers.

Furthermore to protect each motor from being damaged, an external power supply for each motor has been used. Each power supply has been set to limit the current to 400 mA. This value has been determined as the amount of current when the motor's efficiency peaks as shown in the motor datasheet (appendix F).

Circuit Schematic

Appendix B shows a schematic diagram of the final version of the control circuit. Because I have needed to drive robotic worms with up to four segments, the control circuit has four L298 motor drivers, however if the control circuit needs to be extended to control more segments,

5.2 Software

5.2.1 Development Environment

As discussed in the Design chapter of this report, the GUI control application is developed in Java. To develop the application, Netbeans IDE software was used. This software is available for free [**?**].

To develop control code for the Arduino board, Arduino IDE software was used. This software is available for free [2].

5.2.2 Testing Controllers

To enable testing and debugging of the control circuit, two simple open loop testing controllers have been developed in the Arduino code. These controllers do not feature current feedback and control the segments only based on pre-programmed timer. Each segment is set to close and open for the same amount of time. However when a segment is closing, it has to push against the mesh (which works as a spring) therefore to achieve the same closing and opening time, a segment has to be closed with more force then opened. To provide this force speed is set higher for closing, which causes the driver to provide more voltage to the motor.

To determine the speed with which each segment should be closing and opening, each segment has been tested and timed individually and the code has been adjusted accordingly.

The testing controllers do not communicate with a PC (except for start and stop commands), and all computation happens in the Arduino board. To start

or stop the controller Serial Monitor within the Arduino IDE has been used. This Serial Monitor allows sending and receiving commands between Arduino IDE running on a PC and the Arduino board.

The first controller uses the same movement pattern as an inchworm. It first in sequence closes one segment after another, after all segments are closed, it starts opening one segment after another. This pattern then loops, which should propel the worm forward. Using the movement pattern syntax as designed in the Design chapter of this report, this pattern can be written as:

```
C X X X O X X X
X C X X X O X X
X X C X X X O X
X X X C X X X O
```

The second controller uses a two-segment overlap pattern. This pattern closes the first segment, and as it starts opening it, the second segment starts closing, as the second segment starts opening, the third segment closes etc. This pattern can be written as:

```
C O X X X
X C O X X
X X C O X
X X X C O
```

The control circuit has been tested with these controllers and it did function as it was supposed to. The worm did move, however in both cases the worm has not been propelled forward, but rather backwards.

I believe that the backward movement is caused by the fact that both of the controllers are not using any kind of current feedback and use only different speeds to compensate for the spring constant of the mesh when closing. This results in the segment going much faster when closing than opening, which means that the worm has more kinetic energy when closing, which might cause to move backwards rather than forward.

As these testing controllers were not developed as part of the final con-

troller, but rather just a way of testing the control circuit, I have not investigated this issue any further.

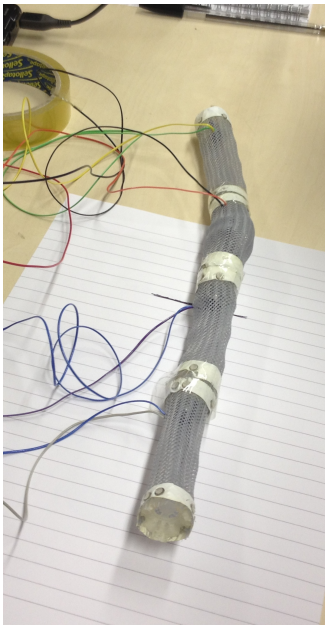


Figure 5.3: Testing of the control circuit with testing controllers.

5.2.3 Communication Protocol

Once the control circuit has been finished and tested, a communication protocol has been developed. This protocol works as outlined in the Design chapter of this report.

Arduino

The Arduino part of the communication has been quite simple to implement, as the Arduino programming language already offers a Serial port interface. A simple handler, which parses all received messages over Serial, has been implemented and tested.

Using the communication protocol following commands were implemented for the Arduino board:

1. **[init]**

   This command serves as a way for the control application to confirm that

it is indeed connected to Arduino driving the robot worm. Upon receiving this command, the Arduino board will reply with "[segments:X]", where X is the number of segments available to the control circuit.

2. **[manual:250>1,100;0,0;]**

   This command tells the Arduino to manually turn the motors of the segments on. The first number specifies the number of milliseconds for which the motors should be turned on, then for each motor a set of two numbers is specified - one setting the direction (0 or 1), the other setting the speed (0 to 100). After this command has finished its execution, Arduino sends "[manual:finished]" back to the computer.

3. **[update:1,100;0,0;]**

   This command updates the current speeds of the motors. When the control application is automatically opening or closing any segment, the control application will keep sending speed updates to the Arduino board every couple of milliseconds. The Arduino board will then change the motor speeds accordingly. For each motor there is a set of two numbers specified - one setting the direction (0 or 1), the other setting the speed (0 to 100).

In case the control application sends a speed update to the Arduino, which turn the motors on, and then the application stops (either because of an error, or because the user has terminated it), the Arduino remembers the time of the last update and if it has been longer than 500 milliseconds, the Arduino will stop to prevent uncontrollable movement.

Lastly, the Arduino board also reads the current feedback every execution of its system loop, if the current feedback is not 0, it sends it to the computer in form of "[feedback:1200>123,45]", where the first number specifies the current time of the reading and then there is a number per each motor specifying the current feedback.

Java Application

Once the Arduino was able to receive commands a Java counterpart needed to be implemented.

While Java does not offer a direct way of communicating over Serial port, there is a free jSSC library available under GNU Lesser GPL license [10].

The Java counterpart of the communication protocol has been implemented using this library.

The communication between a Java application and Arduino has been tested and it proved reliable and fast enough to respond to changes of the system.

5.3 Worm Control Application

After successfully implementing and testing the communication protocol, the focus of the development has changed to the GUI control application written in Java.

5.3.1 Control Application Structure

The structure of the control application has been divided into several packages:

- Package “data” contains data storage and processing classes. Code in this package is responsible for processing the current feedback as well as storing the information about the state of the worm and the application.
- Classes in package ”serial” are responsible for communicating with Arduino.
- Package “ui” contains all classes responsible for interfacing with the user.
- Package “ui.graphs” contains classes responsible for displaying the graphs about the current state of the system.

5.3.2 Closed Loop Controller

Noise Filtering

Following the analysis of current feedback and noise filtering, I have created a simple way of storing and filtering the feedback signal. I have implemented a finite impulse response filter [8]. This filter remembers N last samples and for each new sample returns a sum of all remembered samples each multiplied by a predefined constant. Equation 5.1 formally described this filter.

$$y[n] = \sum_{i=o}^{N} b_i \cdot x[n-i] \tag{5.1}$$

However after some testing the filtering has proven to provide good enough results with filter of size $N = 15$ and all constants equal (which in practice produces same results as a regular running average for the last 15 samples).

To further reduce the noise in the feedback signal, additional oversampling has been implemented in the Arduino code. Instead of simply sending the current feedback to the control application, the Arduino reads the feedback several times for each motor and then sends the average. After some testing the best compromise between noise reduction and speed of the feedback reading has been determined as oversampling of an average from 5 samples.

To summarise the current feedback therefore gets filtered twice - once on the Arduino as the reading is oversampled and second time in the control application, using a running average filter of last 15 samples.

The current feedback samples and the filter are stored separately for each segment (in class "WormController"). The class "WormController" receives all readings from the Serial communication, divides these readings and forwards each sample to its "SegmentController". The segment model then processes the sample (by filtering in) and stores it.

Segement State Recognition

As outlined in the Design chapter, when a segment is supposed to be closed or opened, its segment model is activated and it will keep the motor of the segment

running, until it finds a local maximum or minimum in the current feedback, which represents that the segment has been fully closed (local maximum) or fully opened (local minimum).

To find these local maxima and minima, the active segment model computes the gradient of the current feedback. If the gradient is close to 1, it means that the current feedback has reached either a local maximum or a minimum. The figure 5.4 shows a control digram for the

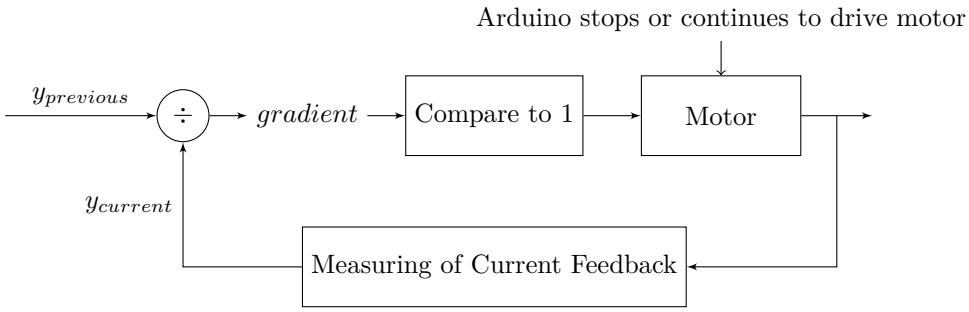


Figure 5.4: Control Diagram for Segment State Recognition

However even after oversampling and filtering the feedback signal, there is still some noise present in the signal. To make sure that the control algorithm is not too sensitive and does not react to small peaks in the signal, which do not represent a local minimum or a local maximum, the feedback sample has to meet the termination condition (the gradient being equal or close to 1) for several readings in row.

For example if a segment is opening, its load is decreasing, which causes the current feedback to decrease as well. This means that its gradient is smaller than 1 and is approaching 1 as the segment is opening. If there is a small peak in the signal, the control algorithm will evaluate the gradient as close to 1 for one or two samples, but because then the signal will continue to descend, the gradient will again fall down bellow 1. Enforcing to meet the condition at least 10 or so times means that the control algorithm will not be over sensitive and will ignore small peaks in the signal. The sensitivity of a segment can therefore be configured by changing the number of times by which the terminating condition is met.

This approach however further delays the signal; therefore a good balance between the sensitivity and the delay has to be found. Because each of the segments of the robotic worm, with which this control application has been tested, is hand made, no two segments are exactly the same. This means that the "sensitivity" for each segment has to be configured individually.

The user can change this sensitivity for each segment in the settings of the control application.

Speed Control

I have implemented a PID speed controller following the description of a PID control in the Design chapter.

To enable the user to use the control application with different robotic worms and to enable them to test with different rates of change of speed, the GUI of the control application contains forms, which enable the user to set the PID parameters to arbitrary values as well as to set dynamic equations for input and setpoint of the PID.

As described in the Design chapter of this report, the input and setpoint equations get evaluated for every single PID computation. While this can be used to set the PID setpoint to a constant value, which then behaves as a usual PID controller, it can also be set to a dynamic value, which changes based on the received current feedback. The user guide of the control application (appendix A) details what variables are available for the dynamic equations as well as explains how to setup the PID properly.

To evaluate the dynamic equations Java Expression Language (JEXL)library has been used, which is available under Apache License [9]. JEXL is a computation engine, developed by The Apache Software Foundation, which provides an API for evaluating dynamic equations.

To provide an easy starting point for the user, a default PID setting has been provided. This setting sets the input equation as the gradient of the current feedback and the setpoint to constant of 1. The PID gains for these equations have been found by interactive tuning by hand.

5.3.3 Segment Control

The control application provides a simple graphic interface for the user to select a single or all segments, and then initiate one of the following actions:

- Manually move the segment(s). This sends a [**manual: ...**] command to Arduino (see the implementation of communication protocol on Arduino above), which manually moves the motors of the segments accordingly to the received command.
- Automatically close or open the segment(s). This uses the above described closed loop controller to control the segments.

5.3.4 Pattern Control

Following the design of the pattern control from the previous chapter, a mechanism for loading user defined movement patterns has been implemented.

The mechanism uses Java provided APIs to prompt the user for a file selection, and then it validates and parses this file. If the file does not pass the validation, the user is informed why it was not possible to load the pattern.

After loading and parsing the pattern, this pattern is saved for each segment individually.

When the user starts the pattern movement, the control application creates an action queue, to which actions from the pattern are added. The control algorithm then takes first set of actions from the queue and executes them on all segments. Each individual action of opening or closing a segment is controlled by the closed loop controller as decribed above. If a segment is meant to be idle, no action is taken. After actions for all segments are finished, the control algorithm takes another set of actions from the queue and executes them. This is repeated until the queue is empty. If the pattern is set to repeat, at that point the queue is again filled with actions from the pattern. The user can pause and restart the movement whenever they choose to.

5.3.5 System State Visualisation

To implement data graphs I have made use of plotting library for Java XChart [6] that is released under Apache License, which allows for free use of this library in the project.

I have used the XChart library to plot both movement pattern graphs as well as current feedback graphs. Both of the types of graphs include sub-plots for each segment to display information about all segments in one graph. XChart provides several predefined visual themes for the generated graphs. A MatLab theme has been used, which makes the generated graphs look very similar to graphs generated in MatLab.

In addition, the current feedback graph also updates in real-time as the control application is receiving most up to date current feedback from the Arduino board.

Furthermore, a worm state graph has been implemented. This graph shows the expected state of the worm, with visualisation of each segment either being closed or opened. The visualisation also shows if a segment is being closed, opened or is idle.

All of these graphs have been implemented as a separate window and by default they are not visible to the user. The user can easily open them if they choose to, either by using a keyboard shortcut or by selecting an option in the menu. This provides the user with a benefit of being able to display just the information the user is interested in. The windows with the graphs can be resized and the graphs can be saved as an image by right clicking on the graph.

Lastly, it is possible to save all recorded current feedback as a comma separated value file (CSV), which can then be processed by other software, such as MatLab or Excel.

5.4 Hardware and Software Testing

Due to the fact that this project was being developed following the spiral process model both the hardware and software implementation of the system have been tested as they were being developed.

When developing a software product, it is usually common to test the functionality by implementing unit tests, which test small parts of the software automatically. This method of testing however is not practical for software, which is closely coupled with hardware, as it is not feasible to automate hardware tests as well as software tests. A mock version of the hardware can be simulated in software to enable the automated tests, however this was not within the time scope of this project. As a results, all testing has been done manually.

Chapter 6

Experimental Pattern Tests

One of the objectives of this project was to utilise the developed control application and circuit and experimentally determine what is the fastest control pattern to propel a segment based robotic worm.

This chapter presents the methodic used to run these experiments as well as the results obtained.

6.1 Experiments

To determine the optimal movement pattern, a set of seven patterns has been considered and evaluated.

While it would be possible to test many more patterns, due to the fact that the available robotic worm has only four segments, the possible movement patterns are limited, as well as many of the possible combinations of closing and opening actions would never propel a worm forward. For instance if all segments close at the same time and then open at the same time, the worm does not move; this and many other patterns have not been tested. To address a limited time frame of this project, only patterns that are assumed to propel the robotic worm have been considered.

The patterns were defined as follows:

- **Pattern C1:** The worm starts fully extended, the pattern closes a seg-

ment while simultaneously opens the previous one, then loops.

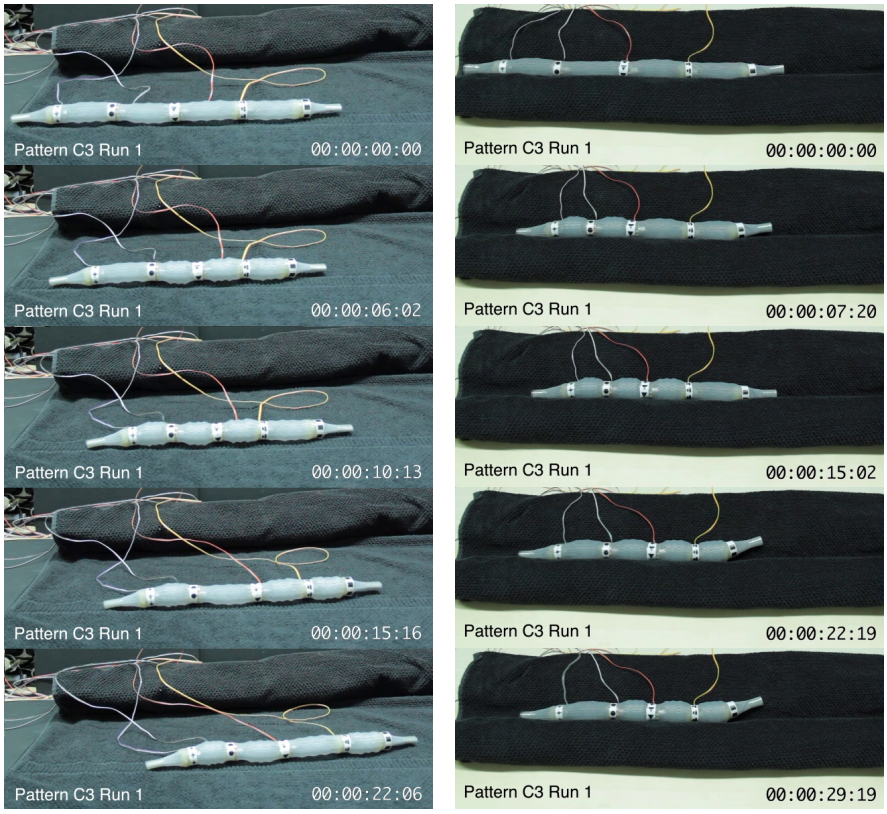
- **Pattern C2:** The worm starts fully extended, the pattern closes a segment while simultaneously opens the second previous one, then loops.
- **Pattern C3:** The worm starts fully extended, the pattern closes a segment while simultaneously opens the third previous one, then loops.
- **Pattern C4:** The worm starts fully extended, the pattern sequentially closes all segments with no overlap; then proceeds to open all segments in the same order. This pattern is similar to the motion of an inchworm.
- **Pattern O1:** The worm starts fully closed, the pattern opens a segment while simultaneously closes the previous one, then loops.
- **Pattern O2:** The worm starts fully closed, the pattern opens a segment while simultaneously closes the second previous one, then loops.
- **Pattern O3:** The worm starts fully closed, the pattern opens a segment while simultaneously closes the third previous one, then loops.

Appendix C shows configurations and control graphs for all of these tested movement patterns.

The speed of the worm has been selected as a criterion for the quality of a movement pattern. Each pattern has been loaded and executed using the control application with current feedback controller, which has been developed as a part of this project. All patterns have been tested on the same robotic worm.

Each of the patterns has been tested on a flat surface and in a half-tube shaped both padded with a rough cloth to imitate interior of intestine. Visual marks were attached to the worm body and the motion of the worm was recorded for each pattern. Figure 7.9 shows stills from the recorder videos. Hyperlinks to the videos are listed in appendix D. Image tracking of the recorder videos has been then performed to analyse the motion of the worm.

When developing the control circuit, two open loop testing controllers have been implemented (see the Software section of the Implementation and Testing



(a) (b)

Figure 6.1: Video stills from testing the worm on a flat surface (a) and in a half-tube (b).

	C1	C2	C3	C4	O1	O2	O3
Mean velocity [mm/s]	2.629	6.037	6.211	2.506	4.178	5.849	1.504
Mean frequency [Hz]	0.482	0.485	0.318	0.151	0.381	0.343	0.230

Table 6.1: Results for flat surface.

	C1	C2	C3	C4	O1	O2	O3
Mean velocity [mm/s]	-0.435	0.668	2.774	1.570	0.635	0.300	0.300
Mean frequency [Hz]	0.447	0.300	0.269	0.182	0.368	0.316	0.246

Table 6.2: Results for a half-tube.

chapter). These controllers use patterns C1 and C4. To provide comparison of the closed loop controller and the open loop controllers, both of these open loop testing controllers have been also evaluated and recorded.

6.2 Results

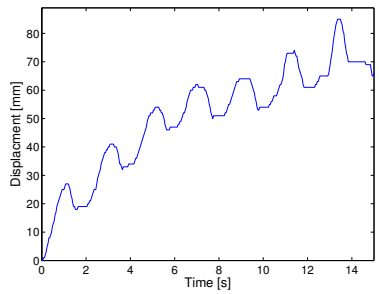
Figures 6.2 and 6.3 show plotted displacement of the rear part of the worm robot for all tested patterns on both flat surface and in a half-tube.

Table 6.1 summarises the motion of the worm robot with tested patterns on a flat surface and table 6.2 summarises the motion of the worm robot with tested patterns in a half-tube.

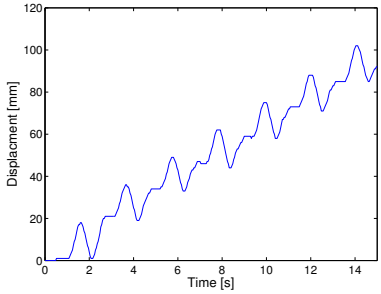
It can be seen that, although the most of the patterns behave differently on a flat surface and in a half-tube, the pattern C3 outperforms all other patterns in both environments. The pattern C2 performs very well on a flat surface, but does not provide much speed in a half-tube. The only pattern, which did not propel the worm forward, is pattern C1, which in a half-tube actually moved the worm backwards 2.5 cm within a time frame of 15 seconds.

The pattern C3 performs the best likely because it provides a good compromise between closed and opened segments at any given time. While there is always enough closed segments to provide good friction, there is also enough segments moving (opening or closing), which actually causes the worm to utilise the friction and move.

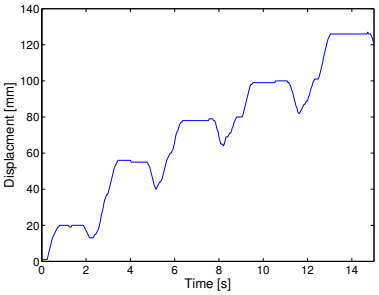
It was observed that all tested patterns have performed better on a flat



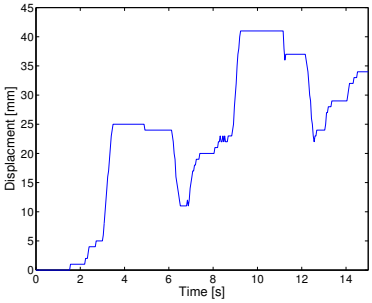
(a) Pattern C1



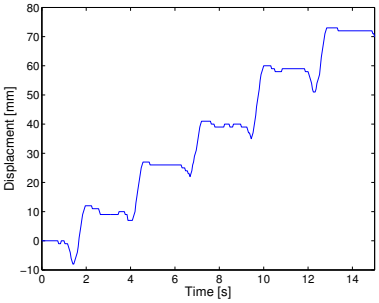
(b) Pattern C2



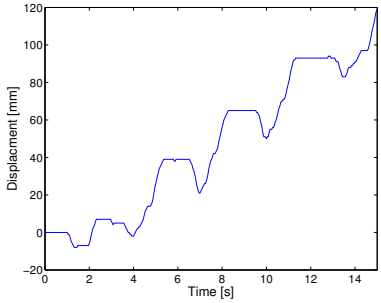
(c) Pattern C3



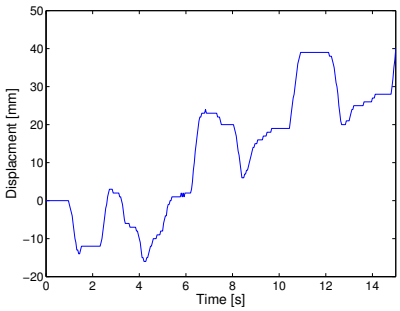
(d) Pattern C4



(e) Pattern O1

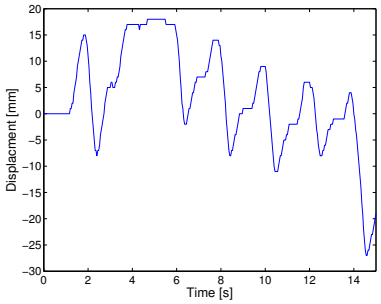


(f) Pattern O2

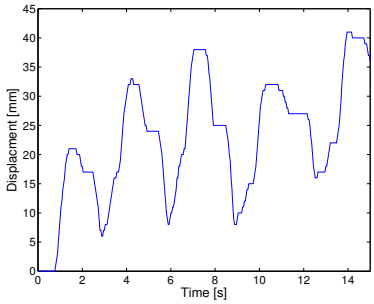


(g) Pattern O3

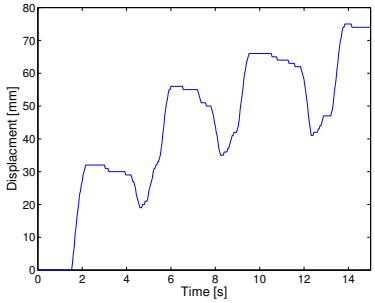
Figure 6.2: The motion of the rear part of the worm robot on a flat surface.



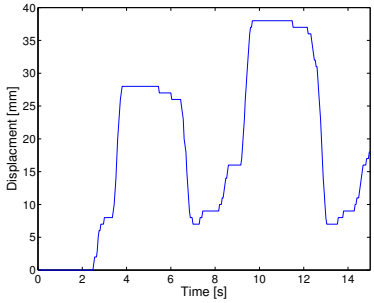
(a) Pattern C1



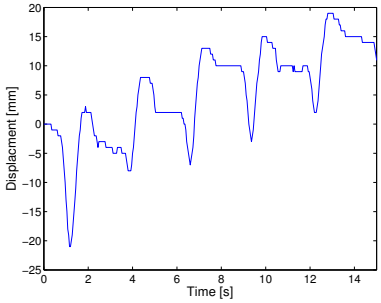
(b) Pattern C2



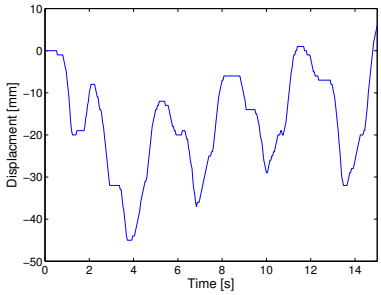
(c) Pattern C3



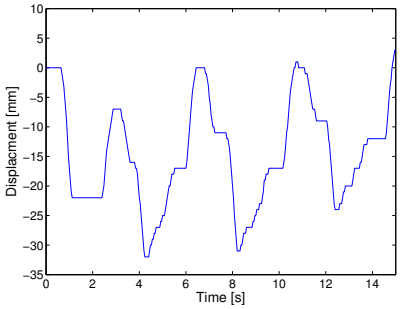
(d) Pattern C4



(e) Pattern O1



(f) Pattern O2



(g) Pattern O3

Figure 6.3: The motion of the rear part of the worm robot in a half-tube.

	C1	C4
Mean velocity [mm/s]	-2.508	-0.1671
Mean frequency [Hz]	0.244	0.166

Table 6.3: Results for the open loop controllers

surface than in a half-tube. This is likely because of the cloth, which has been used to pad both of the testing environments. The purpose of the cloth was to provide an intestine analog and increase the friction to enable the worm to move more easily. However the friction in the half-tube was probably too high and the worm robot was a result more restricted. Nevertheless since both testing environments have been identical for all tested patterns, it is valid to compare them against each other.

6.2.1 Comparison with Open Loop Controller

The open loop controllers have been tested only on a flat surface. Table 6.3 shows the summary of the motion of the rear part of the worm robot on a flat surface.

The results show that both of the open loop controllers do not propel the worm body forward. As discussed in the Implementation chapter, to compensate for the spring constant of the mesh of the worm, the open loop controllers close the segment much faster than they open it, which might cause the worm actually slide backwards, due to the difference in kinetic energy.

While the results of the open loop controllers do not compare to the current feedback controller, they illustrate that unlike an open look controller, a feedback loop controller can adjust to the environment and can perform with better results.

Chapter 7

Evaluation

This chapter will make an evaluation against the requirements of the system set earlier in this report.

7.1 Requirements

To check that the system has been developed accordingly to what has been proposed at the beginning of this project, it must be tested against the requirements outlined in the Requirements chapter of this report. The requirements list describes what the system must be able to do. In this section each requirement will considered against the implemented system to determine if it has been fulfilled.

For each user requirement there is a corresponding functional requirement, therefore only functional and non-functional requirements will be addressed.

7.1.1 Functional Requirements

The functional requirements state that the system must:

1. ***Be able to connect to a segment based worm-like robot with DC motors.***

   Figure 7.1 shows screenshot of the process of connecting to a robotic worm. This requirement has been fulfilled.

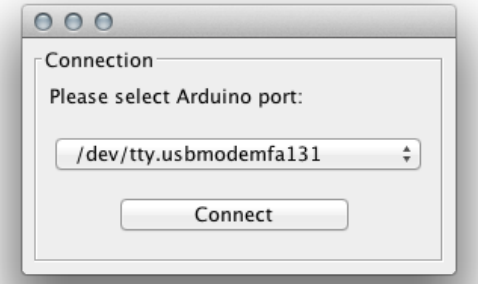


Figure 7.1: Connection window enables the user to select the serial port.

2. ***Provide the user with a GUI to interact with the system.***

   The developed control application provides the user with GUI. Figure 7.2 shows screenshot of this application. This requirement has been fulfilled.

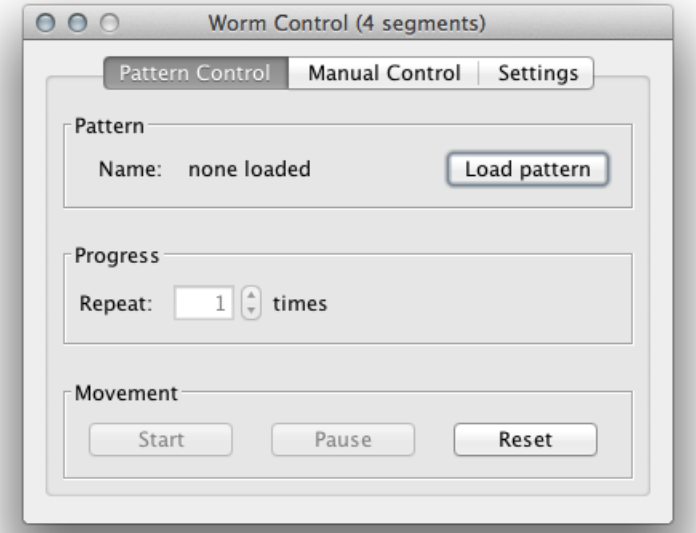


Figure 7.2: Main window of the control application.

3. ***Enable to load a movement pattern configuration and automate the execution of it.***

   Figure 7.3 shows screenshots of the control application loading a movement pattern and chapter Experimental Pattern Tests of this report demonstrates that it is possible to run these patterns. This requirement has been fulfilled.

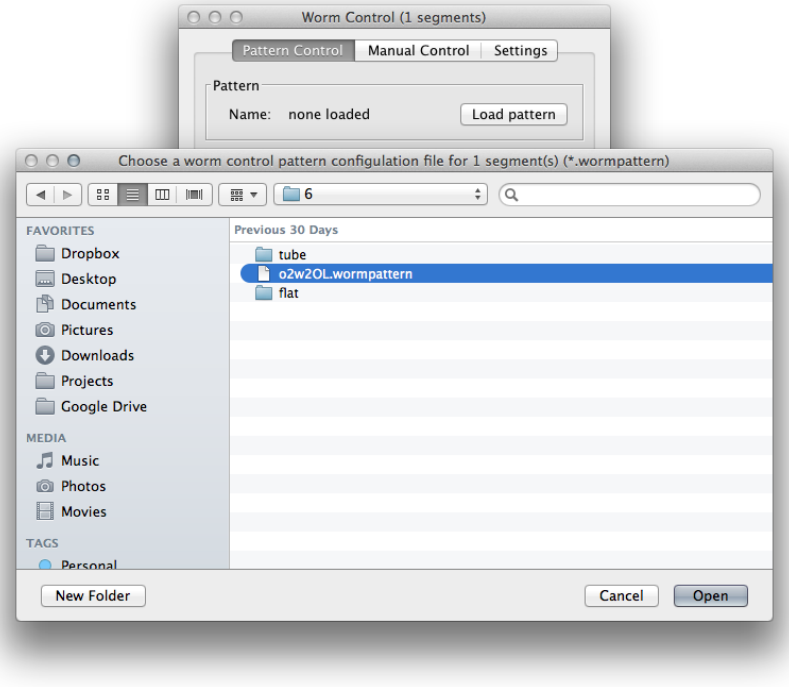


Figure 7.3: Screenshot of a file chooser dialog to load a movement pattern.

4. ***Provide the user with a way of controlling individual segments.***

   Figure 7.4 shows screenshot of the manual control tab of the control application. This requirement has been fulfilled.

5. ***Enable the user to start and stop the worm on request.***

   fig. 7.5 shows screenshot of the control application after a movement pattern is loaded. The user can start and stop the movement of the worm from this screen. This requirement has been fulfilled.

6. ***Produce a data visualisation of the current state of the worm***

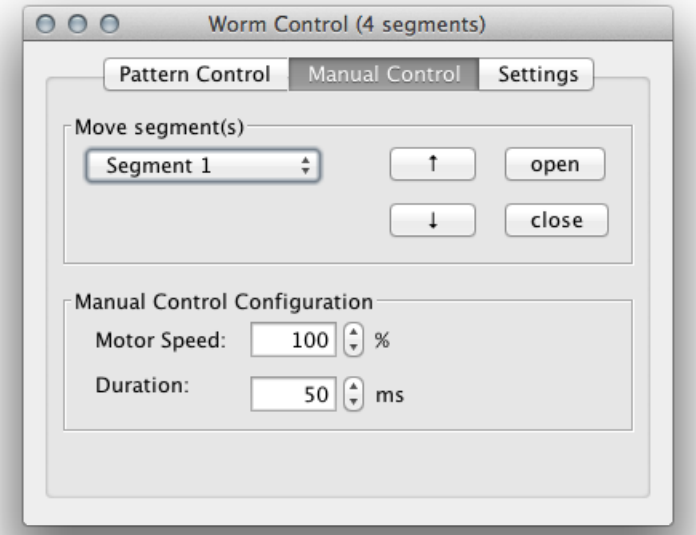


Figure 7.4: Main control window with Manual control selected

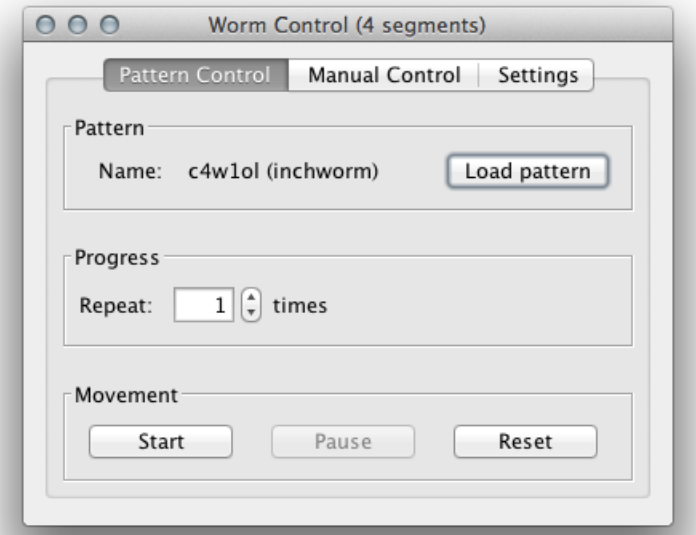


Figure 7.5: Patern control window after a pattern has been loaded

to provide the user with a mean to compare different movement patterns.

Figures 7.6 to 7.8 show screenshots of the control application displaying the current feedback data, worm state visualisation as well as movement pattern graph. This requirement has been fulfilled.

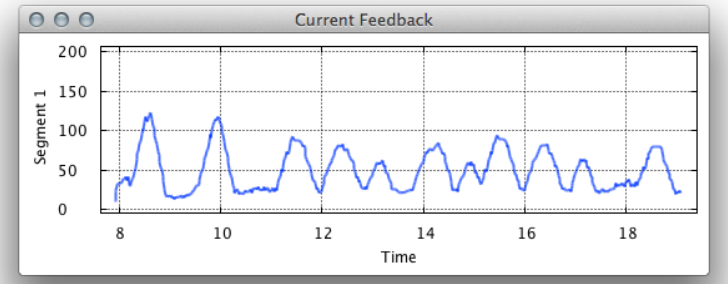


Figure 7.6: Window with Current Feedback Graph for 1 segment.

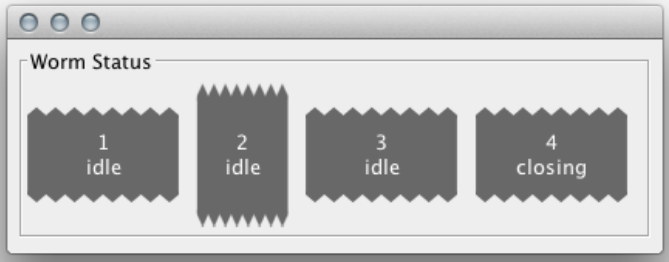


Figure 7.7: Worm status window.

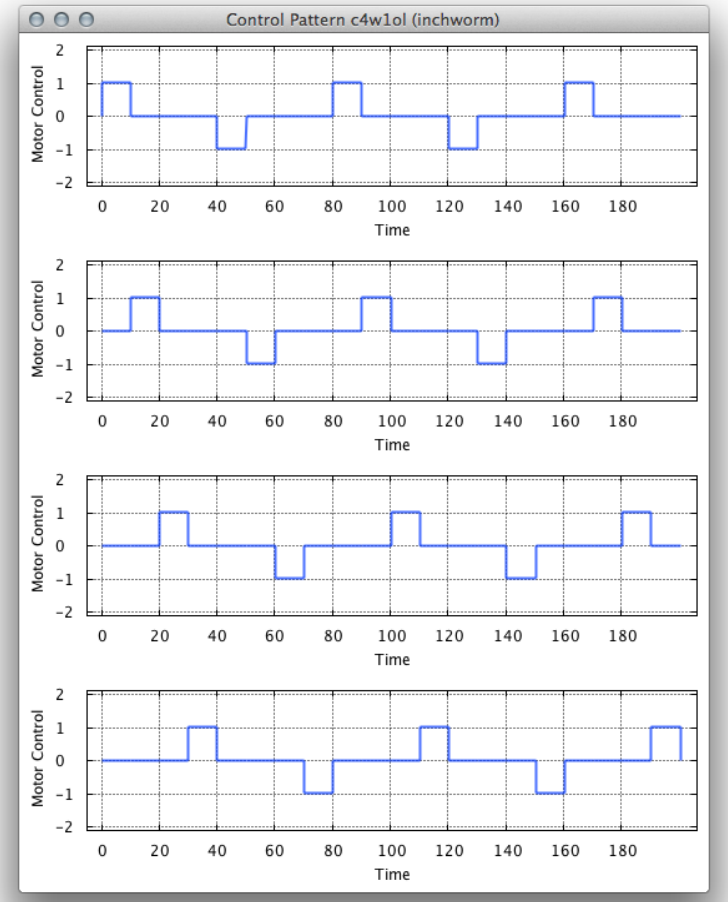


Figure 7.8: Pattern control graph settings window.

7.1.2 Non-Functional Requirements

The non-functional requirements state that:

1. ***The system should be reliable. It should be able to replicate the same results with the same movement patterns.***

   When performing the pattern testing (as described in the Pattern Testing chapter), each pattern has been executed several times, each time yielding the same movement of the worm. This requirement has been fulfilled.

2. ***The system should be responsive. When the user makes an action, the system should either execute such action or provide***

the user with feedback to explain why it is not possible.

While it is not always possible to execute an action right away as the system might be waiting until the worm finishes a current action, the control application has been written to utilise Java threads [1], which makes sure that the UI is always responsive and does not freeze until the action is carried out. This requirement has been fulfilled.

3. ***The size of the system should be minimised to enable portability of the system.***

   When designing the system, importance has been placed on minimising the size of the control circuit. When presented with choice, the smaller component has been used. While the final presented control circuit has been built on a breadboard and therefore is not particularly small, the provided schematic (appendix B) can be constructed in a relatively small size, with the Arduino Mega board being the only limiting factor. This requirement has been fulfilled.

4. ***The user interface of the system should be easy and intuitive to use.***

   When developing the control application several different versions of GUI have been tested. In the effort to simplify the UI, least frequently used functions have been moved from the main window to a menu. These menus can be easily accessed if needed and each action has a keyboard shortcut assigned to it. While this is highly subjective, I believe that the interface of the application is intuitive and easy to use, however to determine if this is really the case further user testing would need to be carried out. This is not possible due to the limited time frame of this project.

5. ***The software part of the system should be able to run on most common operating systems.***

   The control application has been written in Java 8 and utilises the stan-

[1] http://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html

dard Java API [2]. All other required libraries (XChart, jEXL and jSCC) have been included in the source code and the compiled version of the software and as such the application should run on all PC operating systems with Java 8 installed [3]. Figures 7.9a and 7.9b show screenshots of the control application running in Mac OSX and in Windows 7 operating systems. This requirement has been fulfilled.

6. ***The design and implementation of the system must meet the highest professional and ethical standards, which include the ones addressed by the British Computer Society Code of Conduct*** [4]**.**

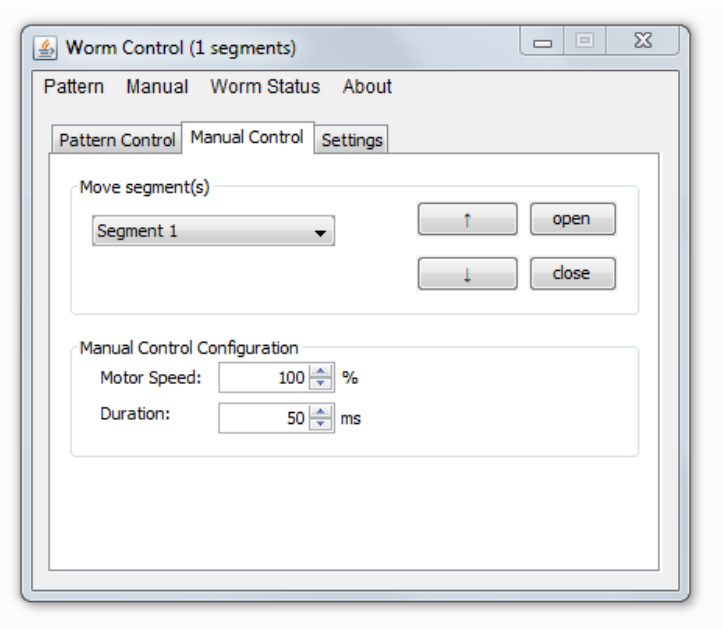
   This requirement has several aspects. Firstly I am the sole author of the design and the implementation of the application including the source code, and all images used in the program, with the exceptions of the declared third party libraries (XChart, jEXL and jSCC) and the used development platforms (Java and Arduino). All of those are publicly available for free use under different licenses. Documents with these licenses have been included with the libraries in the source code of this project. Secondly, the application does not access or save any data to the computer unless the user has authorised such action, with the exception of saving position of each window when it is closed. The window position is being saved to Java Preferences [5], which enables an application to save very small data between launches of the application. It is a common practice that for ease of use, the application saves its state when being closed, so that the user can resume their activity when they launch the application next time. The storage capacity required by this feature is absolutely negligible, and if the user removes the application, the operating system will eventually remove this data as well. Overall I have put very much effort in making sure that the application is developed with

[2] http://docs.oracle.com/javase/8/docs/api/

[3] available at http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html

[4] http://www.bcs.org/category/6030

[5] http://docs.oracle.com/javase/8/docs/technotes/guides/preferences/index.html



(a)

(b)

Figure 7.9: Screenshots of the control application running in Windows 7 (a) and Mac OSX (b).

as high professional and ethical standards as possible.

7.2 Requirements Conclusion

All the defined requirements have been met, or in case of the subjective ones, maximum possible effort has been invested into making the system as good as possible.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The aim of this project has been to create a control platform to simplify testing of segment based worm robots. By designing and building a control circuit and developing a control application with current feedback loop such platform has been created.

The created platform provides a good foundation for testing segment-based robots, but it has been only tested with one model of a segment based robotic worm. It is likely that to enable testing of another models, the system would have to be modified to address the differences in the builds of the robots.

The implemented current feedback controller has proven to be able to reliably determine, when a segment is closed or opened, but unfortunately no way of determing the exact position of a segment at any given time has been found.

Seven different movement patterns have been tested and evaluated and the pattern C3 (see appendix C) has been determined as an overall fastest pattern to propel the worm. However these 7 patterns do not cover all possible

movement pattern combinations.

In comparison with the existing research of peristaltic locomotion in robotics, this project presents a novel systematic way of testing different movement patterns. It serves as a proof of concept for testing of multiple movement patterns and can be further developed and implemented for new robotic worm models.

Using the developed control platform or reimplementing the presented concept of testing multiple movement patterns for further research would enable much broader and more systematic comparison of the efficiency of different movement patterns and robotic worm models.

Overall the project has been successful in meeting its objectives and requirements. However its development is certainly not complete, as many possible features could be added to enhance the functionality and the universality of the control platform. These suggestions for future work are presented in the next section.

Throughout this project, I have gained a lot of knowledge about robotic worms, peristaltic locomotion and soft robotics in general. I believe that segment based worm robots have great potential and could hugely improve the current medical endoscopy procedures.

8.2 Suggestions for Future Work

While the basic concept and the structure of the control application and the control circuit would not have to change, to address the issues or shortcomings mentioned in the previous section, several features could be added or changed.

The implementation of the system has only one type of feedback – current feedback. This is due to the design of the tested robot worm. However in general it is a good idea to have several types of feedback, which increases the speed and resolution of a controller, and also in case of an unexpected problem, the controller can fallback to another controller. To provide more universal platform, a generic controller should be implemented, making it easier to "plug-in" a new robotic worm with new type of feedback.

To address the fact that the control system has been only tested with one robotic worm model, a several different robotic worms should be either obtained or build based on published research and the control circuit and application should be modified and tested to accommodate for the differences. To even further improve the system, a research in different interfaces of (worm) robots should be carried out and generic changes to the system should be implemented to enable support for the most common interfaces.

To improve and expand the results of the experimental part of this project, a worm robot with many more segments should be build and more different patterns should be tested. While the presented results provide a good insight into the study of different movement pattern of segment based worm robots, since earthworms found in the nature have many more segments, by performing such experiments on a robot worm with increased number of segment, much more interesting results could be obtained.

References

[1] Arduino Mega 2560. website. *http://arduino.cc/en/Main/arduinoBoardMega2560*, accessed on 15 April 2014.

[2] Arduino. website. *http://www.arduino.cc*, accessed on 15 April 2014.

[3] Karl J. Åström and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2010.

[4] Soft autonomous robot inches along like an earthworm. website. *http://newsoffice.mit.edu/2012/autonomous-earthworm-robot-0810*, accessed on 15 April 2014.

[5] P. Dario and C.A. Mosse. Review of locomotion techniques for robotic colonoscopy. *IEEE International Conference on Robotics and Automation*, 2003.

[6] XChart. Basic Charts for Java Applications. website. *http://xeiam.com/xchart*, accessed on 15 April 2014.

[7] J. GRAY and H. W. LISSMANN. Locomotory reflexes in the earthworm. *The Journal Experimental Biology*, 15:506–517, 1938.

[8] Finite impulse response. website. *http://en.wikipedia.org/wiki/Finite_impulse_response*, accessed on 15 April 2014.

[9] Java Expression Language (JEXL). website. *http://commons.apache.org/proper/commons-jexl/*, accessed on 15 April 2014.

[10] jSSC java serial port communication library. website. *https://code.google.com/p/java-simple-serial-connector/*, accessed on 15 April 2014.

[11] Byungkyu Kim, Hun-young Lim, Kyoung-dae Kim, Younkoo Jeong, and Jong-oh Park. A locomotive mechanism for a robotic colonoscope. *IEEWRSJ Intl. Conference on Intelligent Robots and Systems*, 2002.

[12] Earthworm locomotion. website. *http://goo.gl/dVVORc*, accessed on 15 April 2014.

[13] Thomas Manwell. Desing of a locamotive surgical worm robot. Master's thesis, King's College London, 2013.

[14] A. Menciassi, J.H. Park, S. Lee, S. Gorini, and P. Dario. Robotic solutions and mechanisms for a semi-autonomous endoscope. *IEEE/RSJ International Conference on Intelligent Robots and System*, 2002.

[15] Raspberry Pi. website. *http://www.raspberrypi.org*, accessed on 15 April 2014.

[16] Serial port. website. *http://en.wikipedia.org/wiki/Serial_port*, accessed on 15 April 2014.

[17] Arduino Motor Shield. website. *http://arduino.cc/en/Main/ArduinoMotorShieldR3*, accessed on 15 April 2014.