Learning with ALiCE II

by
Daniel Lockery

A Thesis
submitted to the Faculty of Graduate Studies of
the University of Manitoba
in partial fulfilment of the requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba R3T 5V6 Canada

Learning with ALiCE II

by
Daniel Lockery

A Thesis submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfilment of the requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Abstract

The problem considered in this thesis is the development of an autonomous prototype robot capable of gathering sensory information from its environment allowing it to provide feedback on the condition of specific targets to aid in maintenance of hydro equipment. The context for the solution to this problem is based on the power grid environment operated by the local hydro utility. The intent is to monitor power line structures by travelling along skywire located at the top of towers, providing a view of everything beneath it including, for example, insulators, conductors, and towers. The contribution of this thesis is a novel robot design with the potential to prevent hazardous situations and the use of rough coverage feedback modified reinforcement learning algorithms to establish behaviours.

Keywords: Reinforcement Learning, line crawling robot, target tracking, monocular vision, rough sets, approximation spaces, ethology

Acknowledgements

Many people have been a great help to me along the way and it has been my privilege to work with them all. My advisor, Dr. J.F. Peters has been a fantastic source of encouragement throughout this adventure and helped make it all possible. Many thanks to both Maciej Borkowski and Christopher Henry from the CILab, who have been excellent to work with and their help has been immeasurable. I appreciated the attention to detail provided by the ECE machine shop when machining parts and taking our orders even during their busy season. Thanks are also extended to the ECE tech shop for their support with parts and helpful suggestions. Also, the work of Marco Peluso for machining short run jobs to help make quick modifications was invaluable throughout the design process. I would also like to express gratitude for the generous support of my research by Manitoba Hydro, and for the advice and suggestions from my advisory committee, including Dr. R. Fazel and Dr. S. Balakrishnan as well as suggestions from Dr. D.S. Gunderson. Also, a special thanks to Chad MacDonald for his helpful suggestions along the way and finally I would like to thank my friends and family for continued encouragement and their support throughout.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Introduction

Current methods for monitoring power grid health consist mainly of sending individuals out to routinely examine different sections. This is generally achieved through one of two ways, walking underneath structures or use of a helicopter and live-line crews to gather top view information. In both cases, digital photography is the method of capturing details for further examination to determine if preventative maintenance is required. Since the Manitoba power grid is vast in size, this is a time-intensive and potentially expensive task. Efforts have been made in the past to design a line crawling robot capable of automating the inspection task, but a prototype was not completed [11, 46]. The work covered in this thesis includes a novel idea for a line crawling robot from design to implementation.

1.2 Motivation

Since monitoring of power grid health is such a complex task, it is difficult to keep track of developing problems, resulting in potential power failure and the need for reactive maintenance. Costs associated with manpower in-

volved in manual monitoring of power transmission lines are significant as is the alternative of outages and reactive maintenance to fix problems as they occur. With automated inspection using a low-cost robotic system, the same area could be covered, but on a continual basis helping to prevent outages through supporting proactive maintenance via inspection. In turn, automation could potentially eliminate the need for manual inspection, reducing costs and ventures into potentially hazardous environments for hydro employees. With the development of a line crawling robot, I plan to show how it is possible to automate the monitoring process.

1.3 Goals and Objectives

The objective of this research is to describe the design and operation of a line crawling robot that is capable of automating power grid inspection. This will be achieved by designing, building and testing a robot following system constraints. A safe testing environment will be developed to simulate inactive power lines for experimental work. Sample targets will be selected to exercise different tracking methods that employ reinforcement learning algorithms to acquire best possible images of targets. The target images will then be stored for later analysis by either a base station or human operator.

The results will be gathered and compared to discover which methods are more robust and most suited for the inspection process.

1.4 Scope

The scope of this thesis covers the robot design, implementation and testing in a simulated environment. Target tracking experiments using a monocular vision system with a couple of tracking methods and an array of learning algorithms are effected to discover the most efficient and best performer for the autonomous design.

1.5 Organization of Report

This report is divided up into seven sections. The second chapter contains the necessary background information required to understand the concepts involved in the work. The third chapter includes a discussion of the system architecture, choices made and potential alternatives where applicable. The fourth chapter discusses system verification and implementation of hardware and software systems. Followed by chapter 5 experimental design, which includes the experimental setup and associated choices made in developing test vectors to facilitate comparison. The sixth chapter contains

the results of the experimental work and a discussion based on the findings. The final chapter concludes the written report and elaborates on the possible direction of future work and expansion for the project.

2 Background

2.1 Line Crawling Environment

Although the ALiCE II robot (Automated Line Crawling Equipment, version 2) is a prototype design, it is intended to operate in a potentially harsh outdoor environment. The line crawling robot as its name implies was designed to crawl along power lines whilst gathering sensory information to report on the health of the power grid around it. Rather than using the conductors as pathways for travelling, an alternative was found in the form of skywire at the top of the towers, also known as a ground wire since it acts like a surge arrestor to protect the power lines.

The skywire is a normally inactive (with exception during surge conditions) wire that presents a much friendlier environment for a line crawling robot to navigate along. There are no concerns of interfering with live conductors or experiencing the harsh environment that they encounter. The example tower structures that were used for the basis of this study are found along the Bishop Grandin Boulevard stretch from Pembina Highway to Lagimodiere Boulevard in southern Winnipeg.

Figure 2.1: Photograph of skywire (top) and conductor (below)

2.2 ALiCE II Systems

The ALiCE II robot is composed of several systems that required some additional theoretical background information. These include sections on gears and motor selection, power supply design, interfacing of processor boards and timing considerations. Each of these topics are addressed in detail for the context of a line crawling robot.

2.2.1 Gears and Motor Selection

The transfer of momentum from the motor drive to the wheels of the robot was accomplished through a gearing arrangement. There were a number of

Symbol	Description
L	Maximum safe tangential load at the pitch diameter in pounds
S	Allowable unit stress for the material in psi
F	Face width of a gear in inches
Y	Outline factor
F_{app}	Applied force necessary to move a given load
F_f	Frictional force
F_w	Force due to the weight of the load being driven
r	Wheel radius
ma	Force due to acceleration of gravity
V	Velocity
F_n	The normal force to the load
μ	The coefficient of friction

Table 1: Symbol descriptions for section 2.2.1

different choices available when examining the possibilities for what type of arrangement to use. A discussion on the selection criteria can be found in the next chapter on system architecture in Sec. 3.2.1. Gear selection was based on the following theoretical considerations.

For proper meshing of the components, the characteristics of both the drive and follower gears were selected with the same profiles. This included the diametral pitch, the number of teeth and the pressure angle. All of these factors help contribute to preventing undue wear, excessive noise or premature breakdown [9]. The diametral pitch is derived from the pitch circle as shown in Fig.2.2. The value for the diametral pitch is represented by the number of teeth per unit measurement of the circumference of the pitch circle diameter [54]. The pitch circle is an imaginary circle that goes through

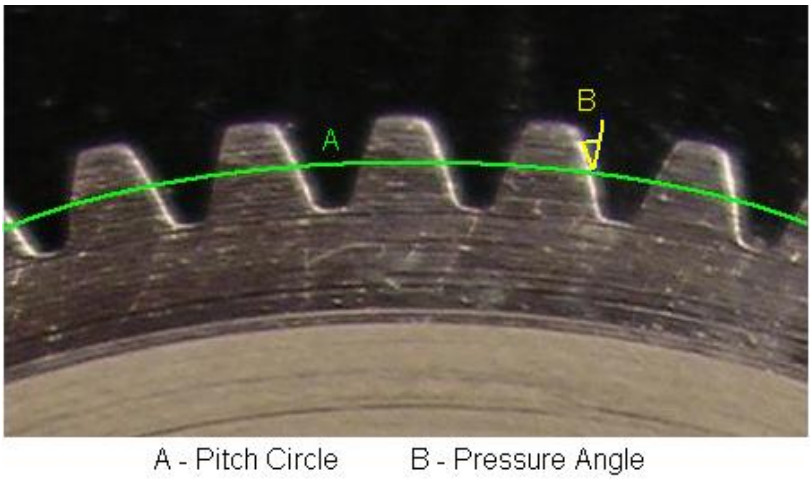


Figure 2.2: Gear profile image

the contact point where gears mesh with each other (approximately the centre of the diameter of the gear teeth) [9]. The pressure angle corresponds to the slope of the gear teeth as seen in Fig. 2.2. The gear tooth profile is slightly rounded on both sides for proper meshing in the forward and reverse directions. The pressure angle is measured between the gear tooth and a line perpendicular from the pitch circle [9]. The number of teeth per gear can differ, but the diametral pitch and the pressure angle must be as close to the same as possible to ensure proper meshing between gears [9].

Another important consideration when specifying gears is to derive the maximum amount of force that can be applied to them safely. For the case of the line crawler, this would occur upon contact with obstacles when the

motor is still driving toward the obstacle before any evasive actions occur. Let *S,F,Y, and P* denote stress, face width, outline factor, and diametral pitch respectively. Further, let L denote maximum safe tangential load at the pitch diameter in pounds. Using Lewis's formula (Eq. 2.1), it is possible to approximate the maximum force that the gears will be subjected to [50].

$$L = \frac{S \cdot F \cdot Y}{P} \tag{2.1}$$

The maximum static torque that a gear can withstand uses the tangential load at the pitch diameter (L), the number of teeth and the diametral pitch of the gear; see Eq. 2.2.

$$StaticTorque = \frac{L \cdot \frac{\#Teeth}{Pitch}}{2} \tag{2.2}$$

Although the gear theory presented here is far from complete, it is sufficient to select the necessary parts for the design to ensure safe operation.

Next, the necessary background for motor selection of the locomotion drive is included. The basic model can be seen in Fig. 2.3. The acting forces on the line crawling robot are shown helping to establish the driving force required to move. The wheels navigate along skywire which can be treated similarly to rolling on smooth terrain including varied angles of inclination

and declination.

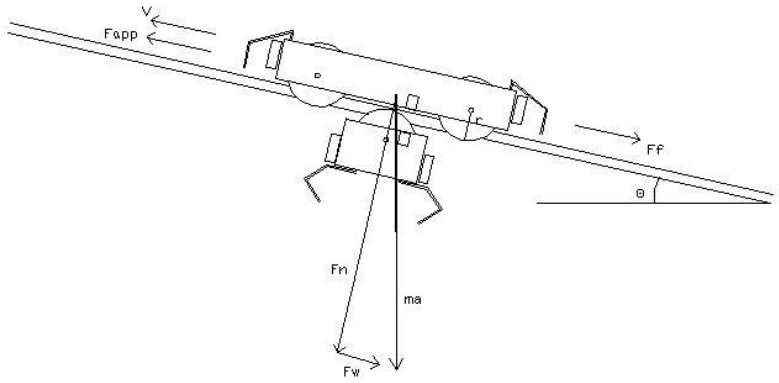


Figure 2.3: Force model for line crawling robot

The components in Fig. 2.3 consist of V, the velocity, F_{app} is the applied force, r is the wheel radius, F_n is the normal force, ma is acceleration due to gravity, F_w is the weight causing the line crawler to slip down a given incline on the sky wire, and F_f which corresponds to the frictional force. The applied force to move the robot is defined in Eq. 2.3, composed of two parts, the frictional force and the weight causing the robot to slip on the sky wire. Overcoming these two forces will result in movement in the direction of choice.

$$F_{app} > F_f + F_w \tag{2.3}$$

The two forces can be represented separately by their own equations [9].

$$F_f = \mu \cdot ma \cdot cos(\theta) \tag{2.4}$$

$$F_w = ma \cdot sin(\theta) \tag{2.5}$$

The angle θ refers to the angle of inclination of the sky wire that the line crawler is travelling on. The term μ corresponds to the coefficient of friction that is usually a small value ranging from 0 to 1 [9].

There are a couple of intermediary steps when moving from the original force model to discovering the required output power and torque of the motor. First, multiplying the applied force and velocity together results in the output power necessary to drive the load, as seen in Eq. 2.6

$$P = F_{app} \cdot V \tag{2.6}$$

The angular velocity can be derived from the velocity as shown in Eq. 2.7, thus providing the necessary components for deriving the required torque from the drive.

$$\omega = \frac{V}{r} \tag{2.7}$$

With all of the previous steps, we arrive at the final equation (Eq. 2.8) for discovering the necessary torque to overcome the forces acting on the line crawler, resulting in locomotion [9].

$$T = \frac{P}{\omega} \tag{2.8}$$

Using the model discussed here for specifying a locomotion motor drive required some design decisions for specifying a few of the unknowns including angle of inclination and the coefficient of friction. These terms and the reasons behind the selections are discussed further in Sec. 3.8.1.

2.2.2 Buck Switching Regulator

Symbol	Description
$V1$	Voltage drop across the Schottky diode, D1
$Vout$	Regulated output voltage
L	Inductance value in Henrys
i	Current measured in amperes
t_{on}	Time that transistor T1 is turned on
t_{off}	Time that transistor T1 is turned off
T	Period associated with switching operation of transistor T1
D	Duty cycle of switching regulator

Table 2: Symbol descriptions for section 2.2.2

A buck switching regulator was chosen to supply the required voltages for the various sub-systems contained within the ALiCE II robot. The buck configuration is a common dc-dc voltage converter that provides regulated

dc voltage at its output from a given input source (batteries in this case). A simplified schematic of the circuit design is provided in Fig. 2.4. A brief

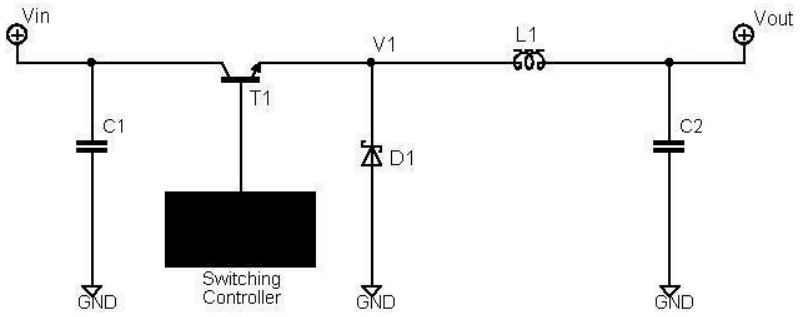


Figure 2.4: Simple buck regulator configuration

overview of circuit operation follows. When power is applied to the circuit and the transistor (T1) turns on, the inductor current increases. After the transistor turns off, stored current will flow through the inductor but instead through the loop containing the Schottky diode. This provides an input duty cycle which yields a desired average output value. This also implies that for the case of the robot's power supply, a continuous current mode design was used (the current never drops to zero). For the buck configuration in Fig. 2.4, a standard input voltage range will provide a fixed, regulated output voltage based on design parameters.

The buck regulator design takes advantage of a switched input voltage, meaning that it has a duty cycle and is on for only part of the time during one

cycle. The duty cycle of the input voltage dictates the output voltage level. To demonstrate how this works, analysis of the inductor voltage follows. First, the expression for the voltage across the inductor, $L1$ is shown in Eq. 2.9 [27].

$$V1 - Vout = L \cdot \frac{di}{dt} \tag{2.9}$$

Next, isolate the rate of change in current, di, yielding Eq. 2.10.

$$di = \int_{ON} (V1 - Vout)dt + \int_{OFF} (V1 - Vout)dt \tag{2.10}$$

The value of L is constant and assumed to be 1 for simplicity. To further simplify the analysis, it is assumed that there is no voltage dropped across the transistor T1 when it is on and also that it exhibits an ideal switching profile [27]. The result is V1 equals Vin when T1 is turned on and Vout is zero when T1 turns off [27]. Adding the upper and lower bounds relating to the duty cycle leads to Eq. 2.11.

$$0 = di = \int_{0}^{t_{on}} (Vin - Vout) \cdot dt + \int_{t_{on}}^{t_{on}+t_{off}} (-Vout)dt \tag{2.11}$$

This can be readily simplified to:

$$0 = (Vin - Vout) \cdot t_{on} - Vout \cdot t_{off} \tag{2.12}$$

Since t_{on} + t_{off} is equal to one full period, denoted as T, this expression can be further simplified to:

$$\frac{Vout}{Vin} = \frac{t_{on}}{T} \tag{2.13}$$

The duty cycle is equivalent to the amount of time that a signal is on during divided by the total period. The right hand side of Eq. 2.13 corresponds to the duty cycle, also referred to as the letter D.

$$D = \frac{Vout}{Vin} \tag{2.14}$$

From Eq. 2.14 it can be seen that the output voltage of the circuit is equal to the input voltage (Vin) multiplied by the duty cycle. Adjusting the duty cycle directly affects the output voltage of the buck switching regulator.

Practical dc-dc converters often have more complicated controllers built into pre-packaged chips or SOT packages using a power MOSFET that allows for variable duty cycles and improved performance over the traditional configuration shown in Fig. 2.4 [58]. Further discussion of the design parameters as well as its associated pros and cons can be found in the system architecture chapter (see Sec. 3.9).

2.2.3 Interfacing Processors

As the project evolved, an additional controller was added to accommodate the increased amount of peripherals and tasks that were needed to operate the ALiCE II robot. An extra controller introduced the problem of interfacing and timing issues for bidirectional communication. After investigating a few possible communication schemes, the most established and reliable method for both controllers was the RS-232 serial data transmission standard.

The RS-232 protocol has been around since the 1960's [64] and is well documented. However, the amount of detail covered in this section provides just enough information to demonstrate what was needed so that the reader can see the advantages and why it is suited to ALiCE II's communication protocol. The first thing to note is the standard pinout for the RS232 cables. The inter-processor communication used a reduced version of the DB-9 connector, shown in Fig. 2.5, which has the pins labelled. Using this setup, all of the remaining handshaking signals were ignored. The intent behind this was to speed up communication to ensure as close to real time processing as possible for critical operations. Keeping this in mind, I selected a bit-rate

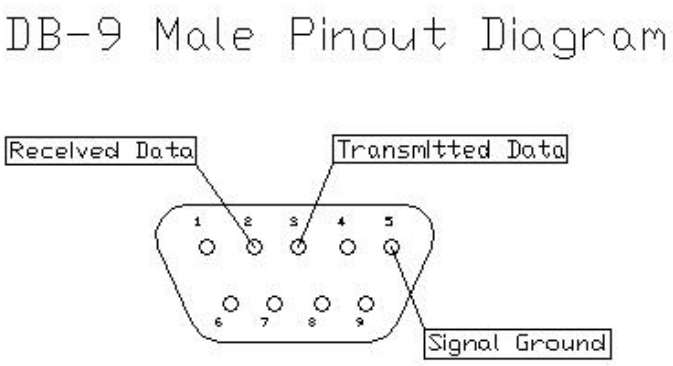


Figure 2.5: Reduced pin set used for simple RS232 communication

with the smallest percentage of error to help reduce the problem of incorrect data packets.

This is a bi-directional scheme that allows communications to be initiated from either processor. The standard provides some flexibility in choice of how data is transmitted, but a common configuration is 8-N-1 [64]. This refers to the number of bits, whether parity-checking is being used and the number of stop bits, respectively. The flexibility of packet size depends upon the number of bits included and whether parity or stop bits are used. Fig. 2.6 shows the protocol pattern that was used, including a start and stop bit as well as eight bits of data. Thus for each 8-bit data transfer there will be one start bit and one stop bit, for a total of 10 bits transferred. Another point of interest with the RS232 protocol is that the logic and voltage levels

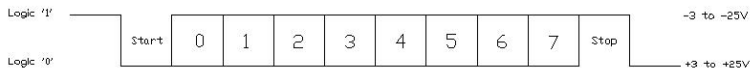


Figure 2.6: RS232 8-N-1 Protocol

are counter-intuitive. A logic 1 is represented by negative voltage levels and a logic 0 is represented by positive voltage levels. Also, the voltage values are represented with non-standard logic, ranging from +/- 3-25 volts for the receiver and +/- 5-15 volts for the transmitter.

The knowledge of signal voltage levels and the communication protocol allows a good understanding of the timing constraints for inter-processor communication, helping select a desirable bit rate and physical configuration for interfacing processors. The configuration and choices made are discussed further in the system architecture chapter, Sec. 3.6 and Sec. 3.7.

2.3 Reinforcement Learning

Reinforcement learning (RL) is the technique used in this thesis to help the ALiCE II robot learn desired behaviours for achieving its goals. For the context of this report, reinforcement learning is applied to an agent (specifically, the ALiCE II robot) that must learn behaviour through trial and error experiences derived from an array of input sensors in a dynamic environment [23].

The trial and error method has associated rewards and punishment values for each action taken [23]. A benefit of this technique is that the rewards and punishment values can be provided without giving a definite specification of how the tasks or goals are to be accomplished [23] [67]. The goal of this type of learning is to direct the actions taken by the agent toward a desired outcome or a given action dependent upon the current state. This is an iterative procedure with the agent learning by selecting both desirable and un-desirable actions, gathering experience and then basing future decisions in tune with the pre-defined goals or tasks it has been given. Intelligence through reinforcement learning draws parallels to human intelligence after a fashion since it starts with sensory systems, gathering information and then based on those experiences, generating predictions about what will happen in similar circumstances at a later point in time [17] The iterative nature of reinforcement learning follows Eq. 2.15 as a general form of the update rule for each step in refining a policy [67].

$$NewEstimate \longleftarrow OldEstimate + StepSize[Target - OldEstimate] \tag{2.15}$$

To avoid getting stuck continually selecting actions that provide high rewards early on potentially resulting in missing out on selecting even better actions later in time, it is important to balance exploration with exploitation [23], [67], [75]. This allows for the possibility of higher valued actions that could reveal themselves only through exploring or sampling a range of possible actions. Balancing exploration and exploitation of actions for any given state can often be found as part of any reinforcement learning method. The experience gathered by learning algorithms is often broken up into episodes to help simplify the processing stage, however continuous learning is also possible as it is better suited to some types of problems. Some algorithms are more dependent upon episode length as they process information to revise their estimates at the end of each episode. For the work reported in this thesis, the experience can be separated easily into episodes as the nature of the problem is discrete with individual time steps. In addition to simplifying processing for some algorithms, episodes can help monitor performance and intermediate results throughout the learning process. The informal goal of learning methods discussed in this thesis is to maximize the available returns in accordance with a pre-defined set of desirable behaviours corresponding to the agent in its environment.

There are three main parts included in a reinforcement learning scheme, states, actions and rewards. These three components make it possible to describe the state space in terms of a range of possible actions and their associated rewards [75]. The intent of each scheme is to come up with a mapping from states to actions based upon experience and the resulting rewards [23]. The mapping is commonly referred to as discovering a policy (denoted π) for exhibiting a desired behaviour based on pre-programmed goals through selecting actions meriting the greatest rewards [23]. Reinforcement learning differs from many other methods of artificial intelligence by being an unsupervised, experience-based technique. The term unsupervised refers to how the mapping is discovered. As opposed to having a 'supervisor' providing training examples of correct behaviour, agents are left to discover appropriate behaviour through exploration and associated rewards and punishments. This approach to learning more closely approximates experiences seen in every day life (learning by doing) as opposed to supervised learning which more closely approximates learning in a classroom environment. Another benefit of reinforcement learning is that multiple actions can occur that produce the best reward in a given situation and previous experience will dictate which one to select or the fact that either are equally good choices, allowing

both of them to be possible selections [23]. Another important distinction for the learning environment is its dynamic nature. There are two possible extremes, a stationary environment where the rewards remain the same for all time and the opposite where rewards differ based upon a changing environment. For the work reported in this thesis, dynamic environments with non-stationary rewards are of interest since they more closely approximate actual conditions that are expected in the field. For example, when the line crawling robot is travelling along the sky wire in windy conditions or through terrain with variable illumination due to shadows and simultaneously attempting to gather images it needs to be able to compensate for external influences. There will also be periods of calm with a more static environment and the associated rewards will change accordingly. One more point about reinforcement learning that is important in affecting behaviour is the use of delayed rewards [75]. The example contained in [75] is the well known *pole balancing problem* where a pole is supported on a cart that moves in the vertical plane and the pole is hinged on the cart to move in the vertical plane as well. The goal of the problem is to create a policy that allows the cart to move in such a fashion that it counter-balances the pole, keeping it upright at all times. One extra constraint are stoppers at each end

of the track to prohibit unlimited movement in any direction. In this example, it is easy to see that actions taken in the moment can have dramatic effects later on as the cart approaches the end of the track if it is moving too quickly toward the end, the pole will easily topple over and become unbalanced (See Fig. 2.7 for my rendition of the pole balancing problem). With these points in mind, the next step is to include a discussion of the algorithms that employ these principles.

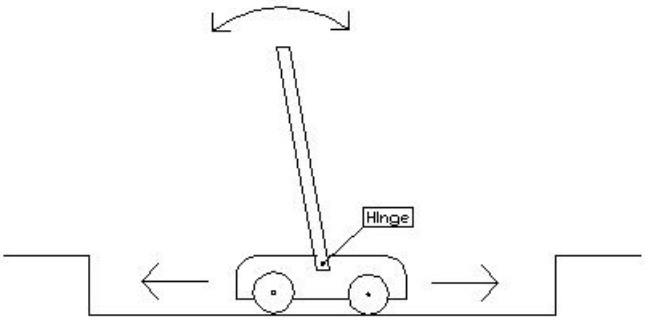


Figure 2.7: Pole balancing problem example

There are a number of different algorithms to implement reinforcement learning. The work reported in this thesis includes a few of them, the actor-critic method, Q-learning, and sarsa. These algorithms provide a mainstream view of temporal difference reinforcement learning. Each algorithm

is discussed in turn, developing the theory and presenting the formal algorithm in the corresponding section. Further discussion of the implementation of each algorithm along with variable parameter selection can be found in the corresponding sections of the system architecture and verification discussions (See Sec. 3.11.1 and 4.3).

2.3.1 The Actor Critic Algorithm

Symbol	Description
$\pi(s,a)$	Policy mapping state to action selection
$p(s,a)$	Preference associated with selecting action, a, given state, s
β	Step size parameter for preference revision
δ	Temporal difference error
r	The reward provided for selecting an action in a given state
γ	Discount factor for weighting future rewards
$V(s_t)$	The value assigned to the current state at step 't'
α	The learning rate step size adjustment parameter

Table 3: Symbol descriptions for section 2.3.1

The actor critic algorithm is composed of two separate parts, an actor and a critic. The main idea stems from a concept called reinforcement comparison [67] where positive actions are encouraged and negative actions are discouraged by adjusting the preference of selecting them for future decisions in a positive and negative manner respectfully [67]. The given policy being followed is referred to as a search element or $actor$ since it is responsible for mapping any given state to an action [65], [67]. The value function

or estimate is labelled the $critic$ element since it is able to quantify the actions taken by the actor and suggest improvements to the policy [65], [67]. Together, the interaction of the policy and the value function make up the actor critic approach to reinforcement learning.

First, the main principles of the actor critic method are discussed before delivering the formal algorithm. As previously mentioned, the actor is the name provided to the policy being followed as it is acting on sensory input from the agent's environment. The policy converts sensory input to a given state which is then mapped accordingly to a desired action. The range of possible actions have preference values associated with their selection [67]. The preference of selecting an action is a helpful way to balance exploration and exploitation since it helps uncover the best possible actions. This balance is achieved through a $softmax$ action selection method which encourages selection of better actions whilst discouraging poor action choices as seen in Eq. 2.16 from [67].

$$\pi(s, a) = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}} \tag{2.16}$$

The preference of any given state action pair is the probability of selecting the action in that state. Through using the $softmax$ selection method,

this helps to ensure that extremely poor actions are unlikely to recur whilst helping positive actions become more likely. This is comparable to human learning in nature, once a really poor outcome is experienced, it is highly unlikely that choices will be made to re-visit that same situation. As seen in Eq. 2.16, the policy values are determined from the preferences for selecting a given action. The idea mentioned earlier about reinforcement comparison also applies here as preferences are modified based on the reward associated with selecting a given action. The adjustment of preference values are made using Eq. 2.17 [67].

$$p(s,a) \longleftarrow p(s,a) + \beta \cdot \delta \tag{2.17}$$

Preference adjustment introduces two new terms, β and δ. The value of β is always positive and it is a step-size parameter that determines the amount of adjustment to the preference value that takes place at any given time step during the learning process [67]. The value of δ is referred to as the temporal difference (TD) error and can be expressed as Eq. 2.18. The TD error provides the critic with a performance measure for generating feedback for

the actor regarding its current policy.

$$\delta = r + \gamma \cdot V(s_{t+1}) - V(s_t) \tag{2.18}$$

Several new terms are introduced in the TD error, including r, γ, and $V(s)$. The first new term, r, constitutes the reward value given in a particular state for taking an action, a. Next is γ referred to as the discount rate. The value of γ is a positive number, generally between 0 and 1 that provides a measure of the current value of a future reward [67]. The result of adjusting γ is as follows, as it approaches zero, the value of the current state becomes more important, taking more of a greedy stance. As the value of γ moves closer to one, future rewards are given more weight. This can be thought of as how some people plan for the future, there are some that want everything right here and right now (γ = 0) and others that put off immediate reward by pursuing an alternate path that provides greater rewards later on (γ closer to 1). Finally, the last new term is $V(s)$, which refers to the value of being in a given state [67]. This value, as seen in the equation refers to future rewards or what can be expected in return for visiting a particular state [67]. Since the actual expected values are unknown for anything but a completely specified model of an agent's environment, the value function is

an estimate [67]. The estimate comes from sampling the next state and adjusting the value function accordingly [67]. This method of updating lends itself well to real time applications with dynamic environments, creating a favourable candidate for the ALiCE II platform. The term α referred to

Algorithm 1: The Actor-Critic Method

Input : States $s \in S$, Actions $a \in A(s)$, Initialize α, γ.
Output: Policy $\pi(s, a)$ responsible for selecting action a in state s.
for $(all\ s \in S, a \in A(s))$ **do**
  $p(s, a) \longleftarrow 0$;
  $\pi(s, a) \longleftarrow \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}}$;
end
while *True* **do**
  Initialize s;
  for $(t = 0; t < T_m; t = t + 1)$ **do**
    Choose a from s using $\pi(s, a)$;
    Take action a, observe r, s';
    $\delta = r + \gamma V(s') - V(s)$;
    $V(s) \longleftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$;
    $p(s, a) \longleftarrow p(s, a) + \beta\delta$;
    $\pi(s, a) \longleftarrow \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}}$;
    $s \longleftarrow s'$;
  end
end

as the learning rate, is used in updating the value estimate controls the step size of the adjustment at each step. This is a useful parameter in specifying how much correction to value estimates occur, small values of α are useful for minor corrections when the policy is performing well and larger values

of α are useful for poorer performance. The value of α can be determined experimentally or set up using prior knowledge, it is commonly selected as a constant value between 0 and 1.

The policy being followed by the actor critic algorithm is considered the actor (π) and the value function provides the critic ($V(s)$) or feedback performance on how the actor is doing at any given time step. Through refining the actor's approach, the critic helps to direct the agent's focus toward the best policy for achieving its goal [48].

2.3.2 The Q-Learning Algorithm

Symbol	Description
$Q(s, a)$	Action value, associated with state
α	The learning rate step size adjustment parameter
r	The reward provided for selecting an action in a given state
γ	Discount factor for weighting future rewards
$\pi(s, a)$	Policy mapping state to action selection

Table 4: Symbol descriptions for section 2.3.2

The Q-learning algorithm presents an alternate approach to the learning problem from the actor-critic method in that it learns based on the action (or Q) value associated with each state as opposed to using the value function associated with being in a given state. Q-learning was developed by Watkins, formally reported in 1989 [75], which is why the algorithm is often

referred to as Watkins' Q-learning. Since there is only a single component (actor) in this method, it is simpler to implement, but it makes use of some different concepts for its operation. This section includes a brief description of the algorithm operation followed by a closer look at the key steps and finally the formal algorithm will be included for completeness.

Q-learning in its simplest form is a single step temporal difference learning method that is capable of maximizing the action value of an agent regardless of the policy being followed [67], [23], [76], [75]. The basis of a single step algorithm is that it looks into the future one step in advance when estimating the best course of action to take from the current state. The best action is generally the choice that maximizes the future discounted reward available from all possible actions pertaining to the current state. Another important point in discussing Q-learning is that it doesn't matter what policy is being followed, it will always maximize the action value [75]. Q-learning falls into the category of off-policy algorithms [67]. This implies that the decisions made for selecting a course of action do not necessarily follow the policy that is exploring the state space [67]. Depending on the situation, it may be advantageous to follow the same policy for both exploring and action selection, but that is discussed further in the next section. An advan-

tage of using the off-policy approach is that the exploration vs. exploitation problem discussed earlier can be addressed directly. The policy can include an element of exploration that allows all states and actions to be explored and at the same time for examining the best course of action, a greedy policy that chooses the maximum discounted reward for the next step ensures the best possible action is subsequently taken. Together they make a strong case for discovering the best policy and there have been several works that detail convergence proofs for Q-learning including [75], [76], and [21].

In understanding the algorithm mechanics, there are a couple of points that will be expanded upon including policy initialization and the update rule for revising the action value. During the initialization of the algorithm, a policy must be established to determine a preliminary (most likely sub-optimal) mapping from states to actions. As listed in the formal algorithm, this policy is not greedy implying that there is a possibility that it will not always follow the action with the highest reward. There must be at least a small chance that this policy will explore alternate actions, providing potential visits to all actions since it is possible they may return increased long term rewards. During the learning process of single step Q-learning, state-action pairs are examined one step ahead. Rather than following the

original non-greedy policy that is selecting actions, a greedy policy is used to determine the best action to take from the current state.

$$Q(s,a) \longleftarrow Q(s,a) + \alpha[r + \gamma \cdot max_{a'} Q(s',a') - Q(s,a)] \tag{2.19}$$

As seen in Eq. 2.19, although the non-greedy policy is selecting actions, the update is affected by the maximizing greedy policy inherent in the Q-learning update rule. The algorithm parameters that can be found in Q-learning are γ, and α which correspond to the discount factor and the learning rate step size adjustment as discussed in the previous section for the actor critic algorithm. Further discussion of the actual values chosen are included in Sec. 4.3. The last step in the discussion of Q-learning is inclusion of the formal algorithm.

One additional point regarding the formal algorithm is the introduction of an $episode$. As previously discussed, this refers to a length or amount of time steps present in an episode of analysis for the reinforcement learning process. In some cases, at the end of an episode, extra processing can take place for performance analysis or to adjust some of the algorithm parameters or possibly to determine when to stop the learning process (especially in static environments). In continuous learning environments however, where

Algorithm 2: The Q-Learning Method

Input : States, $s \in S$, Actions $a \in A(s)$, Initialize Q(s,a), α, γ, π to an arbitrary policy (non-greedy)
Output: Optimal action value Q(s,a) for each state-action pair
while *True* **do**
  for $(i = 0; i \leq \#ofepisodes; i++)$ **do**
    Initialize *s*
    Choose *a* from *s*, using policy derived from Q
    Repeat(for each step of episode):
    Take action a; observe reward, r, and next state, s'
    Q(s,a) $\longleftarrow$ Q(s,a) + $\alpha[r + \gamma max_a Q(s', a') - Q(s, a)]$
    s $\longleftarrow$ s'; a $\longleftarrow$ a';
    until *s* is terminal
  end
end

there are no clearly defined episodes, this can be ignored and the algorithm operates continuously, revising its policy toward the focus indefinitely or until a stopping criteria is met. For the work reported in this thesis, clearly defined episodes were used throughout the learning process.

2.3.3 The Sarsa Algorithm

Symbol	Description
$Q(s, a)$	Action value, associated with state
α	The learning rate step size adjustment parameter
r	The reward provided for selecting an action in a given state
γ	Discount factor for weighting future rewards
$\pi(s, a)$	Policy mapping state to action selection

Table 5: Symbol descriptions for section 2.3.3

The final algorithm included is sarsa, named by Sutton [66], [59]. Origi-

nally, sarsa was treated as a variant of Q-learning and it was first introduced as $modified\,Q-learning$ in the literature by Rummery and Niranjan [56]. A concern with single step Q-learning was that it would select only greedy actions with the nature of its off-policy approach to maximize the next state-action selection, potentially missing out on actions that could provide much improved rewards at a later point in time [56]. The sarsa algorithm was developed out of this concern to improve performance. This section includes a brief discussion of how sarsa differs from Q-learning, as well as the formal algorithm and an overview of how it operates.

The main difference between Q-learning and sarsa is in their approach to updating action values. Unlike Q-learning, sarsa uses an on-policy approach for learning [67]. This implies that the policy that is exploring is also the same policy that is used for updating the action value estimates Q(s,a) [67], [56], [66], [59]. The formal update rule for sarsa is provided in Eq. 2.20.

$$Q(s,a) \longleftarrow Q(s,a) + \alpha \cdot [r + \gamma Q(s',a') - Q(s,a)] \tag{2.20}$$

The difference in the update rule is that the next state-action pair is not maximized for action value as it is in Q-learning. The updated value is instead

drawn from the acting policy $\pi(s, a)$, which selects the next action from the given state following the same policy that is used to explore state space. Often the exploring policy will be somewhat greedy, usually based on a parameter ϵ, also known as $\epsilon - greedy$ policies [67]. This provides some degree of exploration along with a generally greedy trend when selecting actions. The parameter ϵ is often selected through trial and error or from prior knowledge about the operating environment. Lower values favour more static environments where greedy tactics prevail and higher values favour more dynamic environments where some exploration is required to find the best actions. Next, the formal algorithm for sarsa is included (Alg. 3).

Algorithm 3: The Sarsa Method

Input : States, $s \in S$, Actions $a \in A(s)$, Initialize Q(s,a), α, γ, π to an arbitrary policy (non-greedy)
Output: Optimal action value Q(s,a) for each state-action pair
while *True* **do**
  for $(i = 0; i \leq \#ofepisodes; i++)$ **do**
    Initialize *s*
    Choose *a* from *s*, using policy derived from Q
    Repeat(for each step of episode):
    Take action a; observe reward, r, and next state, s'
    Choose action a' from state s' using policy derived from Q
    Q(s,a) $\longleftarrow$ Q(s,a) + $\alpha[r + \gamma Q(s', a') - Q(s, a)]$
    s $\longleftarrow$ s'; a $\longleftarrow$ a';
    until *s* is terminal
  end
end

Sarsa makes use of the policy being followed when selecting an action for the next state during the update process. This policy employed encourages exploration as it is soft or non-greedy and it is used to generate a' for the next state prior to the update rule. Besides the policy followed, updating the action value takes on the same form as Q-learning, using the discounted future reward to adjust the current value. The output of the sarsa algorithm are optimal action value estimates Q(s,a) for each state-action pair. A formal discussion of this convergence for on policy methods can be found in [60]. The last remaining subtle difference is that the next state-action pair will contain whatever action the current policy being followed selected as opposed to the always greedy action. This is how the name sarsa came about as each iteration of the algorithm uses the current $s$tate, $a$ction, $r$eward, and the following s'tate, a'ction pair [66].

2.4 Rough Set Theory

This section of the report provides some background and introduces the details of rough set theory used in conjunction with the reinforcement learning algorithms previously discussed. Rough set theory was introduced by Zdzislaw Pawlak in the early 1980's as a means to deal with data or sets with im-

Symbol	Description
U	A non-empty, finite set of objects
A	A non-empty, finite set of attributes
$IS(U, A)$	An information system composed of (U,A)
V_a	Value(s) associated with attribute a $\in$ A
$\sim_B$	Indiscernibility relationship for X (an IS) over B (B $\subseteq$A)
B^*X	Upper approximation of set X $\subseteq$ U based on attributes B $\subseteq$ A
B_*X	Lower approximation of set X $\subseteq$ U based on attributes B $\subseteq$ A
$BN_B(X)$	Defined boundary area of set X $\subseteq$ U based on attributes B $\subseteq$ A

Table 6: Symbol descriptions for section 2.4

precise boundaries, referred to as rough sets [42]. Since its inception, there have been a number of different fields reporting applications including business, environment, electrical and computer engineering, finance, and medicine to name a few (eg. [44], [78], [2], [41], [19], [74]). An introduction to rough set theory including a simple example to demonstrate the ideas follows.

Rough sets are used as a tool for addressing imprecision, vagueness or unknowns in data with the goal being to uncover patterns, rules or knowledge [25]. The philosophy behind rough sets is that they treat every object in the universe of discourse as an item with attributes [44]. The attributes can consist of any feature of the data provided such as colour, shape, thickness or numerical quantities. Representation of objects and attributes occurs in the form of data tables to facilitate the processing steps [47]. A data table

is referred to as an information system (IS) and is composed of a pair, U which refers to a non-empty finite set of objects and A, a non-empty finite set of attributes [25].

$$IS = (U, A) \tag{2.21}$$

Each row in the data table represents an instance of the type of data being examined, for example using a medical data table, each row might represent an individual patient [25]. Every attribute $a \in A$ contains a set of values, V_a referred to as the domain of a [45]. Often an extra column is added to an information system in the form of a decision attribute. This is for applications where an outcome for any instance of the given data set is known [25]. Using an IS that contains decision attribute(s) can be seen as a form of supervised learning since any kind of processing benefits from the prior knowledge of the outcome from each object [25]. A simple IS example including a decision attribute can be seen in Table 7 and is commonly referred to as a decision system (DS) [25]. This is a simple IS with a decision attribute for determining whether to work outside or not depending upon weather conditions. Each row in the table can be treated as an 'If...Then' statement [39] where if the attributes indicate a given pattern, the outcome

x_i	*temp.*	*wind*	*precip.*	*work outside*
$x0$	$20°C$	*light*	*none*	Yes
$x1$	$35°C$	*stormy*	*heavy rain*	No
$x2$	$24°C$	*none*	*none*	Yes
$x3$	$20°C$	*light*	*none*	No
$x4$	$15°C$	*strong*	*light rain*	Yes
$x5$	$8°C$	*none*	*none*	Yes

Table 7: Sample Information System

will be followed. Although this is an extremely simple example of a data table, it is helpful for demonstrating some key concepts.

First, the concept of indiscernibility can be shown from cases $x0$ and $x3$. The attributes for these cases are identical, yet the decision attribute is opposite. This is referred to as an indiscernibility relation and is more formally described by Eq. 2.22.

$$\sim_B = \{(x, x') \in U^2 | \forall a \in B a(x) = a(x')\} \tag{2.22}$$

Where $C = (U, A)$ is an IS and $B \subseteq A$, the two cases for $a(x)$ and $a(x')$ are indiscernible from one another [25] based on the attributes found in B. The example for the case of $x0$ and $x3$ from Table 7 show that using the features, temperature, wind, and precipitation these two instances are indiscernible from one another (formally shown in Eq. 2.23).

$$[x0]_B = \{x0, x3\}, \tag{2.23}$$

where $B = \{temperature, wind, precipitation\}$.

Set approximations provide a distinction of whether a set is roughly definable or crisp. First, let $X \subseteq U$, and include B as a set of attributes for each element in X, giving the pair $IS = (X, B)$. There are two approximations of interest when determining if an instance in the IS belongs to X, the upper and lower set approximations. They are denoted B^*X and B_*X for the upper and lower approximations respectively [47].

$$B^*X = \{x \in U | [x]_B \cap X \neq \emptyset\} \tag{2.24}$$

$$B_*X = \{x \in U | [x]_B \subseteq X\} \tag{2.25}$$

where $[x]_B \in \frac{U}{\sim_B}$ [47]. Using the approximations provided in Eq. 2.24 and 2.25, it can be discovered if a set is crisp or rough. The lower approximation is composed of all members of the set that are definite members of X [25]. The upper approximation is composed of elements that can potentially be members of X [25]. The result of generating these approximations is three separate areas of distinction in the partition of the Universe represented by B. They are members that can be classified with certainty as belonging

to X, those that do not belong to X, and those that possibly belong to X. The upper and lower approximations help define a boundary area which is provided in Eq. 2.26 [68].

$$BN_B(X) = B^*X - B_*X \tag{2.26}$$

The difference between the upper and lower approximations provide the boundary condition and the difference between rough and crisp sets becomes apparent [68]. When the boundary region $BN_B(X)$ is equal to the empty set $\emptyset$, then X is referred to as crisp with respect to B [68]. Alternately, if the boundary region is not empty, then cases exist that can not clearly be defined as a part of X and accordingly the set is referred to as rough.

Using the information from Table 7 as a simple example, it is possible to determine if the set is rough or crisp. Through the use of a simple diagram, it can be seen that this IS is roughly-definable as the instances $x0$ and $x3$ are indiscernible from one another making it impossible to define with certainty if either belongs to X or not. A pictorial representation is shown in Fig. 2.8 including the upper and lower approximations and the boundary region. The boundary $BN_B(X)$ contains more than just the empty set, indicating that the

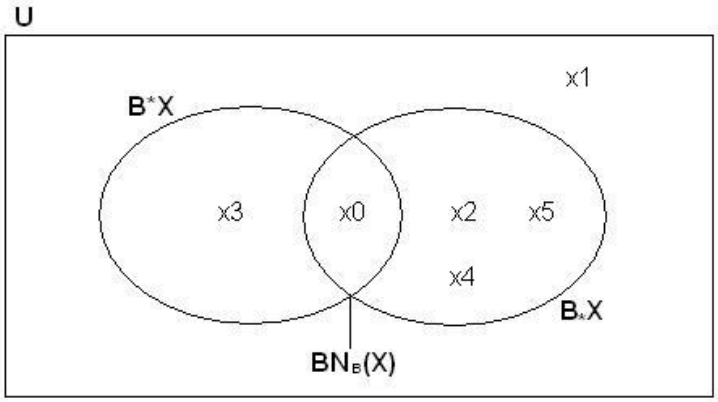


Figure 2.8: Rough set example diagram from table 7

set is not crisply definable.

This concludes the introduction to rough set theory, for more in depth material there are many resources available, including [42], [43], [25]. The next step is to extend the concept of rough sets to include approximation spaces which are part of the application for the work described in this thesis.

2.4.1 Approximation Spaces

Symbol	Description
U	A non-empty, finite set of objects
Ind	Indiscernibility relation on subsets of U
$\mathcal{P}(U)$	The power set of U
I	Uncertainty function for developing blocks of similar instances
ν	Overlap function [0,1] for two subsets of U

Table 8: Symbol descriptions for section 2.4.1

Approximation spaces were originally introduced by Pawlak [42] and have since been expanded to a generalized version. Approximation spaces provide the basis for bridging reinforcement learning and rough set theory together to achieve a common goal of learning acceptable patterns of behaviour in the context of a line crawling robot. This section introduces approximation spaces and the generalized version with rough inclusion and coverage.

The definition of an approximation space provided by Pawlak [42] contained the pair $(U, \sim_B)$. Where U corresponds to a non-empty finite set and $\sim_B$ represents an indiscernibility relation [42]. More recently, a generalized approximation space was introduced by Skowron and Stepaniuk [61], [63] represented by a triple, (U, I, ν).

- U is a non-empty set of objects, and $\mathcal{P}(U)$ is the powerset of U,
- $I : U \rightarrow \mathcal{P}(U)$ is such that $x \in I(x)$ for any $x \in U$,
- ν : $\mathcal{P}$(U) x $\mathcal{P}$(U) $\rightarrow [0, 1]$ is an overlap function (inclusion or coverage).

Similar to the classical description, U corresponds to a non-empty, finite set. The uncertainty function, I provides a neighbourhood for all sample

elements in U [47] such that a given object x is associated with a set of objects that are similar in some respect. This function can also be used to help define a covering of U [47]. Pertaining to coverage of sets, ν is a measure of overlap and is referred to as $inclusion$ or $coverage$ depending upon the configuration of the expression, Eq. 2.27 shows the format for rough inclusion [63].

$$\nu(X,Y) = \begin{cases} \frac{|X \cap Y|}{|X|}, & \text{if} \quad \text{X} \neq \emptyset, \\ 1 \quad , & \text{if} \quad \text{X} = \emptyset. \end{cases} \tag{2.27}$$

Although they are similar, $rough\ coverage$, represented in Eq. 2.28 has the opposite term in the denominator. For the purpose of this paper, $rough\ coverage$ is used as a measure of set overlap to provide a performance metric.

$$\nu(X,Y) = \begin{cases} \frac{|X \cap Y|}{|Y|}, & \text{if} \quad \text{Y} \neq \emptyset, \\ 1 \quad , & \text{if} \quad \text{Y} = \emptyset. \end{cases} \tag{2.28}$$

Both $inclusion$ and $coverage$ provide a means of measuring the overlap for any sets $X, Y \subseteq U$. The value of ν represents the degree of coverage, ranging from 1, when the sets are equal to one another (X = Y) to the minimum value of 0, when there are no common elements in X and Y (X

$\cap\, Y = \emptyset$) [63]. Anything in between 0 and 1 represents at least some degree of overlap between the two sets in question. A brief example is included to demonstrate the key concepts for approximation spaces.

2.4.2 An Example Approximation Space

Symbol	Description
U	A non-empty, finite set of objects
A	A non-empty, finite set of attributes associated with objects in U
d	A decision attribute belonging to any instance
$DS(U, A \cup \{d\})$	Decision system
B	A block of attributes, B$\subseteq$A
D	Decision class, instances where d = 1 (accepted or true value)
B_*D	Lower approximation of set D $\subseteq$ U based on attributes B $\subseteq$ A
ν	Overlap function to determine rough coverage value [0,1]

Table 9: Symbol descriptions for section 2.4.2

In this section, a brief example of an approximation space is presented. The context of the example begins with a decision system modelled after the actual tables used for the line crawler experiments. The data will then be partitioned into subsets or blocks based on attributes. The final step is to take the blocks and measure the degree of coverage with an accepted set of positive observations (the lower approximation). The simple example will provide a big picture of how this idea could be extended to more complicated situations with a greater number of entries and attributes to provide a measurement of system performance.

First, a decision system is included with similar attributes to those used during experimental work with the line crawling robot. The decision sys-

x_i	$state$	$action$	$preference$	$reward$	$decision$
$x0$	1	4	0.37	0.62	1
$x1$	2	7	0.42	0.50	1
$x2$	1	3	0.34	0.16	1
$x3$	1	3	0.34	0.16	0
$x4$	2	7	0.42	0.50	1
$x5$	3	2	0.17	0.18	0
$x6$	3	3	0.24	0.21	1
$x7$	0	4	0.08	0.12	0

Table 10: Decision System for an Approximation Space Example

tem is represented by $(U, A \cup \{d\})$, representing the universe U, the set of attributes for each instance A, and the decision attribute d associated with each instance. The table presented in this section is considered a sample (X) as it represents a very small part of any of the actual tables used under normal conditions. Often when creating a block of attributes, a subset $B \subseteq A$ is chosen. For this example, $B = A$, but for notation, I will refer to the block as B. This implies that the block B = $\{state, action, preference, reward\}$. The next step is to establish the decision class, I selected $d = 1$ as my accepted instances, yielding a decision class shown in Eq. 2.29.

$$D = \{x \in X : d(x) = 1\} = \{x0, x1, x2, x4, x6\} \tag{2.29}$$

Subsequently, blocks are generated based on the indiscernibility relation

between the objects in the decision system. A block, $B(x)$ consists of all elements in the table that have the same attribute values in B. This is formally defined in Eq. 2.30.

$$[x]_B = \{y|a(x) = a(y) \forall a \in B\} \tag{2.30}$$

At this point, each block in the table is defined according to Eq. 2.30.

$B(x0) = \{x0\}$

$B(x1) = \{x1, x4\}$

$B(x2) = \{x2, x3\}$

$B(x5) = \{x5\}$

$B(x6) = \{x6\}$

$B(x7) = \{x7\}$

Even though there are eight instances in the table, two of them have matching instances that are indiscernible, resulting in only six blocks. At this point, the lower approximation is developed (See Eq. 2.25).

$$B_*D = \{x0, x1, x4, x6\} \tag{2.31}$$

The lower approximation represents accepted cases or desired behaviours. The degree of overlap for the instances in the sample table with already known desired outcomes is an important performance metric. This can be measured through rough coverage shown in Eq. 2.28 by substituting in terms from the example, resulting in Eq. 2.32.

$$\nu(B(x), B_*D) = \begin{cases} \frac{|B(x) \cap B_*D|}{|B_*D|}, & \text{if} \quad B_*D \neq \emptyset, \\ 1 \quad , & \text{if} \quad B_*D = \emptyset. \end{cases} \tag{2.32}$$

The rough coverage values provide a means for gauging how similar the current blocks are compared to accepted or existing blocks from the lower approximation. The values for ν demonstrate the degree of overlap for each case.

$\nu(B(x0), B_*D) = 1/4 = 0.25$

$\nu(B(x1), B_*D) = 2/4 = 0.50$

$\nu(B(x2), B_*D) = 0/4 = 0.00$

$\nu(B(x5), B_*D) = 0/4 = 0.00$

$\nu(B(x6), B_*D) = 1/4 = 0.25$

$\nu(B(x7), B_*D) = 0/4 = 0.00$

These results indicate that the behaviours exhibited in blocks 0, 1, and 6 have some degree of overlap with previously known acceptable cases. The remaining blocks have no overlap and are represented by ν values of zero indicating undesirable behaviour. This metric provides performance feedback that is helpful in finding out how well current performance agrees with previously known acceptable cases.

2.5 Image Processing

Symbol	Description
M	The number of rows in a 2D image
N	The number of columns in a 2D image
R	Red component of a pixel, range from 0-255
G	Green component of a pixel, range from 0-255
B	Blue component of a pixel, range from 0-255
I	Grey level intensity, consists of RGB components added and averaged
D	Decimation factor, an integer value

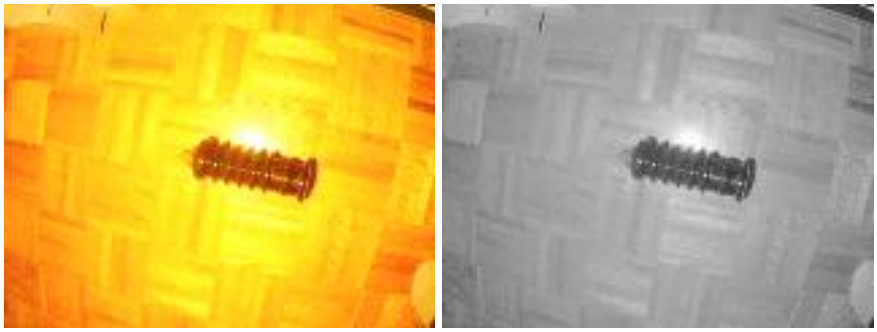
Table 11: Symbol descriptions for section 2.5

This section includes an introduction to image representation, processing operations and the approaches used for tracking targets in conjunction with the line crawling robot.

Representation of images in the context of this work is in two dimensional space. As a result, they can be treated as matrices (of dimensions MxN) [20]. Each element of the matrix represents an individual picture

element or pixel [22]. When dealing with pixels, the origin of the image coincides with the first element in the (0,0) position [22]. The images discussed in this work were all initially captured in colour. For most cases, the human eye contains a group of sensors known as cones and it is these cones that are responsible for colour detection, responding to light at different wavelengths [15]. Generally, these include red, green and blue, which is a triple that coincides with the values of any given pixel [72]. Raw images are sensed and recorded in colour so each pixel is represented by three quantities corresponding to the red, green and blue content [15]. The values of each colour component are represented by an integer, often ranging from 0-255 (represented by 8-bits). The makeup of any given pixel is some combination of these three values to provide a composite colour. The end result is a matrix containing MxN pixels of data with each pixel composed of three elements, an R, G, and B component.

Since the line crawling robot has limited capabilities, image processing is required to reduce the computational cost of gathering information from them. The first measure taken was to convert the image pixel values to grey scale, immediately reducing the amount of data by a factor of three. Conversion of colour images draws parallels to the natural sensitivity of



(a) Colour image (b) Grayscale image

Figure 2.9: Demonstration of grey-scale conversion using Eq. 2.33

the rods in the human eye which vary for the different wavelengths in the order of approximately 65% for red light, 33% for green light, and 2% sensitivity to blue light [15]. For simplicity, Eq. 2.33 was used, giving each colour channel the same weighting. An insulator is shown on a hardwood flooring background to demonstrate the results of using this conversion (See Fig. 2.9). This was sufficient for the working environment presented in this thesis to provide grey scale images from the original RGB components.

$$I = \frac{R + G + B}{3} \tag{2.33}$$

In addition to converting images to grey scale, the size of images were shrunk to help reduce the amount of required processing power. Reduction

of 2D-images works the same way as 1D-signal processing but expanded to two dimensions. This is achieved through spatial $decimation$ along the rows and columns of the matrix representation for an image. $Decimation$ of an image results in a reduced amount of data preserving the larger features but at a cost of losing fine details [52]. The process of $decimation$ is similar to filtering as it keeps only the desired information and rejects the excess [52]. For a two dimensional image using a filtering system, decimation operates by passing multiples of an input based on the $decimation$ factor D and rejecting all other information (See Eq. 2.34).

$$\tilde{f}(x,y) = \sum_{y=0}^{(\#rows/D)-1} \left[\sum_{x=0}^{(\#cols/D)-1} f(xD, yD) \right] \tag{2.34}$$

A small image template of 8x8 pixels in size can be seen in Fig. 2.10 with dark squares representing pixels that are kept and light coloured squares representing pixels that are discarded when using a $decimation$ factor of 2. The reduction in pixels is four times fewer as they are $decimated$ in both the x and y directions, producing a 4x4 image. The loss of detail is evident from looking at Fig. 2.10, but for the context of the work reported in this thesis, the example images are large enough that decimation by a small

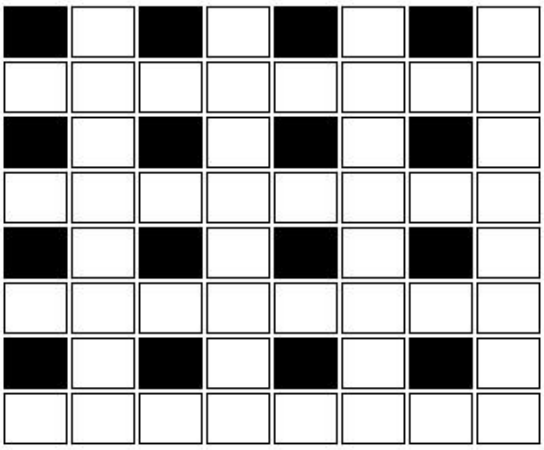
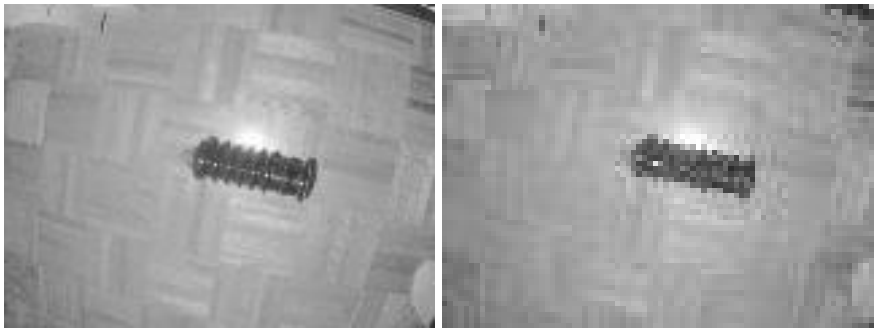


Figure 2.10: 8x8 image template demonstrating decimation by a factor of 2

factor still provides sufficient detail. The sample image seen in Fig. 2.9 has been included with a normal grey scale version of the scene next to a copy of the image $decimated$ by a factor of two. The loss of detail is easily noticed but the coarse features of the image are still well defined and useful for the tracking methods discussed in the next sections.

2.5.1 Template Matching

The first method discussed for target tracking is template matching. The main concept behind this approach is to have a target template and attempt to locate that target within a larger image. This assumes that the image will always be larger than the template. This type of method is similar to what was used by Gaskett in his work with robot navigation when searching for



(a) Greyscale image (b) Decimated by factor of 2

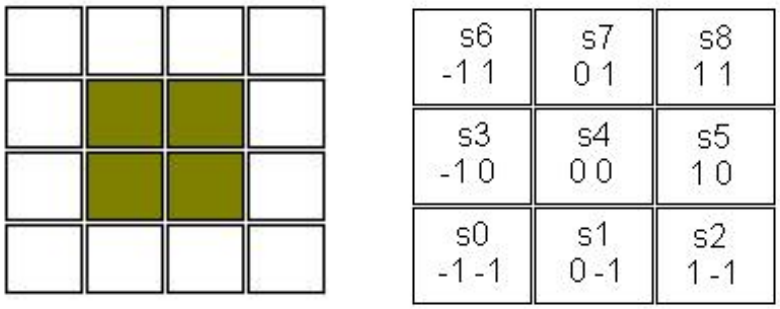
Figure 2.11: Demonstration of decimation by a factor of two

targets [14].

Template matching is fairly straight forward to implement using the sum of squared differences (SSD) as a measure for the best match. The SSD method for template matching uses Eq. 2.35 to derive SSD values for each sub-image of template-size within the original image.

$$SSD(x,y) = \sum_{j}\sum_{k}\left[input(j,k) - template(j-x,k-y)\right]^2 \qquad (2.35)$$

When implementing Eq 2.35, the best match of the template in the target image will occur at the point where the SSD value is lowest. Ideally, a perfect match would result in a value of zero and anything above indicates the target is not contained (or not entirely contained) within the template. Template matching operates similar to correlation but yielding the smallest



(a) Image template (b) template matching states

Figure 2.12: Demonstration of template matching

output for a match instead of the largest respectively [15]. A very small sample image of 4x4 pixels is shown in Fig. 2.12 with a template size of 4 pixels, giving a possible nine template cell matching states. The states are represented by s0 through s8 and the numbers located in each cell indicate the distance from the origin at the centre (0,0).

This method has some advantages and disadvantages that made it suitable for use in preliminary experimental work. First, the main advantages are that this method is fairly straight forward to implement and it provides good results for translational changes. However, there are some disadvantages associated with it as well. The processing power required to perform the SSD method on larger images where many templates fit into each image

can be cumbersome. Also, template matching is not very adaptable to scaling, and sometimes rotational changes depending upon the target in question. For the context of the work in this report, template matching is well suited to smaller sized images with translational tracking and cases where a target (or targets) is previously known and can be referenced.

2.5.2 Average Grey Level Tracking

The average grey level tracking method was implemented with the intent of reducing the processing power required for the tracking problem as well as reducing the effects of scaling issues that thwart template matching. This section of the report includes a brief discussion of the assumptions made, how the technique works, and the potential advantages and disadvantages associated with it.

The average grey level tracking method makes an assumption in locating targets as it searches for the darkest part in its field of view. For the experimental procedures, targets were used that were distinctively dark on a light background. Target images were divided up into a number of subsections and the average grey level for each was generated. The lowest grey level represented the target of interest and in the presence of several low val-

ues, the centre segment was considered the target. The idea stemmed from an attempt to reduce the amount of computation as well as from examining adaptive thresholding techniques [15]. Instead of having a template as seen in the previous approach, each sub-section containing MxN pixels is operated on with Eq. 2.36 to generate its average grey level.

$$AGL = \frac{1}{M \cdot N} \sum_{y=0}^{M-1} \sum_{x=0}^{N-1} f(x, y) \tag{2.36}$$

The section containing the lowest grey level becomes the target or in the event of multiple sections with the same average grey level, the section in the centre of the group is selected. Once a target has been acquired, tracking coordinates are provided to re-position the camera. The current state as seen in Fig. 2.12 dictates the action(s) that need to be taken in order to track the target.

Since the amount of computations were reduced to additions, a single multiplication and division, support for larger images was provided. For example, the state diagram for a 120x160 pixel image is seen in Fig. 2.13, including a larger number of possible states and allowing for finer control based on the improved resolution. There are advantages and disadvantages associated with the average grey level tracking method. The main advantage

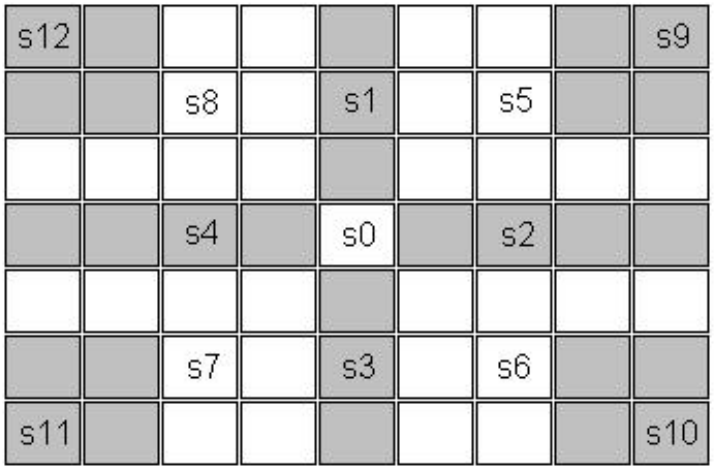


Figure 2.13: Diagram of states for 120x160 pixel image for AGL method

of this method is the reduced amount of computations, resulting in faster computational speed. This also makes it possible to use larger images with higher resolution to help improve accuracy when hunting for targets. The main disadvantage of this method is that the target accuracy is only as good as half the height and width of the size of a sub-section of the image. This is because the targets are not based on individual pixels within the image but the actual sub-sections of images. Comparing this to the previous method, the average grey level tracking method appears faster but also incurs the potential of more error in the targeting process. Increasing the resolution of the images processed is a potential means to offset the disadvantage of the

increased error but at the trade off of more computational requirements for a larger image size.

2.6 Robot Behaviour

The line crawling robot was designed with more than one goal in mind. Besides acquiring useful imagery of power lines and associated equipment, survival value and obstacle avoidance were two other immediate concerns that needed to be addressed. In order to run a robot with multiple simultaneous goals, a layered set of behaviours supported in a hierarchical fashion was helpful. The hierarchy was a key point since depending on the situation, certain behaviours would be more important to deal with than others. The type of architecture that most suited this need was developed by Brooks, referred to as $subsumption\ architecture$ [4] and has been seen in several examples since [5, 6, 10]. This section of the report discusses the details of how this method operates, an example and finishes by addressing its limitations.

The $subsumption$ method was first reported by Brooks in the mid-80's as a control architecture for autonomous robots. His intent was to provide a control scheme that was structured with layers to increase the competence of

robot operation [4]. Each layer specifies a class of behaviours for the robot in question meant for any environment it encounters [4]. The lower layers are reserved for more general behaviours as opposed to the higher layers which are included for more specific goals [4]. Each subsequent layer of competence beyond the first is provided by additional sensory awareness and processing of the additional input to derive the desired behaviour [4]. The idea of multiple goals enters into each layer as well, they all have separate goals and can be worked on nearly simultaneously. The sensory input can be used by all of the different behaviours and often it is not used in the same capacity by each layer [4]. The hierarchy of these behaviours allows the higher level layers to effectively bypass or $sub - sume$ the lower level behaviours, preventing them from driving the outputs to help the higher status behaviours achieve their goals [4]. The lowest layer generally consists of a rudimentary control scheme, exhibiting survival behaviour. This layer is designed and implemented first and once finished, usually remains unchanged even through the addition of subsequent layers [4]. When adding new layers to the behaviour hierarchy, additional processors may need to be added to support them depending upon the level of complexity [4]. The hierarchical architecture is setup in such a way that it easily supports the idea

of an individual processor for each behaviour with minimal communication required between levels [4].

To solidify the concepts, a brief example based on an early design of the line crawling robot is included. The first level of the design (layer 0) included rudimentary control over the motor drive and position servos. The goal for this level was to prevent damaging the robot by avoiding immediate obstacles in the near vicinity during travel along the skywire. Infrared sensors were used to monitor the position of the grip relative to the sky wire and collision detection switches were used to sense objects in the pathway of travel. Together, these types of sensors comprise the layer 0 inputs responsible for defining the obstacle avoidance behaviour. In Fig. 2.14 the connections for sensors and the motors they control can be seen as well as a block for the direction and speed control provided to the locomotion drive. This allows for complete control of robot movement in conjunction with the first layer in the hierarchy.

The next layer of the design had a similar goal of obstacle avoidance but with the addition of avoiding contact with external forces completely. This was achieved through the addition of proximity sensors which use infrared signals to determine when objects are in their field of view and take action

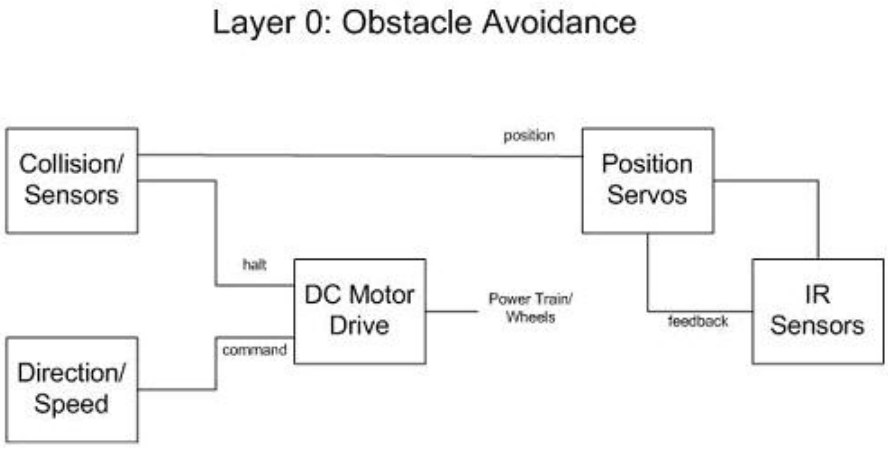


Figure 2.14: Layer 0 control behaviour for avoiding obstacles

accordingly rather than the collision detection switches that require contact before sensing. This layer is able to shut off the behaviour from the lower level if it senses an obstacle in its proximity so it can respond more quickly to achieve its goal of avoiding contact. The diagram in Fig. 2.15 is nearly identical to the previous layer with exception to the introduction of a new block connected to both control lines from the contact switches. This contact point acts like an over-ride for the proximity sensor obstacle avoidance behaviour. When the proximity sensors fire indicating an obstacle is in the field of view, the layer 1 behaviour will take over from layer 0 and direct

the course of action for the line crawling robot, proactively avoiding hard contact with the obstacle. Subsequent layers can be added after this point as

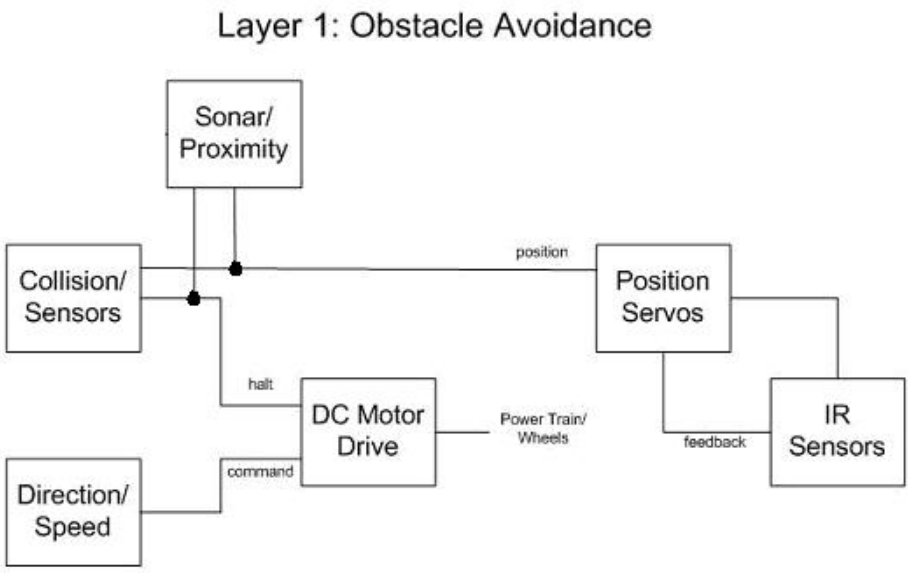


Figure 2.15: Layer 1 control behaviour for avoiding obstacles

needed depending upon the goals of the robot. Besides obstacle avoidance, acquiring useful images is another primary goal of the line crawling robot. An example layer 2 would include a vision system that seeks out targets to take images of and could also be used to determine where obstacles are and map them out for improved obstacle avoidance and cooperative behaviour. During the stages where information from level 2 were used, the other two layers would be suppressed, allowing it to take control.

The $subsumption$ control architecture discussed in this section of the report is convenient for implementing during stages of development. The layers of competence are added with increasing complexity as the project evolves. As a result, the first layer of competence is the most rudimentary and yet essential control, whilst the second, third, and beyond are later additions that provide improved or more sophisticated control. The additional layers are not essential but they are often helpful additions that improve the reliability of the design.

There are a couple of limitations associated with this control method. First, the control layers are fixed once they are in place. This means that each layer provides a reflexive set of responses to given inputs but anything outside of its repertoire during runtime will not necessarily be handled well. Including a learning component along with the fixed responses allows for some flexibility that helps prevent the behaviour constraints once implemented. The second potential problem is for larger systems with more layers and a complex hierarchy. With many more goals and behaviours being active all at once, it is likely that the goals will interfere with one another and there is no way to resolve multiple outputs simultaneously, there needs to be a minute delay between their resolutions. For the example case and

the early stages of the work presented here, the level of sophistication is not great enough to be a cause for concern.

$Subsumption$ is an attractive control scheme for a project during the build stages due to its modularity and simplicity of the hierarchical structure, helping address multiple goals as they are added to the system. However, when moving beyond a simple 'insect-like' intelligence level, an alternate means for behaviour control are necessary [55]. To take this type of architecture one step further, a learning element was added (see Sec. 2.3).

2.7 Ethology and the Ethogram

This section discusses $Ethology$ and the $ethogram$ and how it relates to the work in this report. $Ethology$ is defined as the study of animal behaviour and an $ethogram$ is a table containing a list of the different behaviours that an animal species can exhibit. A brief overview of Ethology is included followed by an explanation of how it parallels the way behaviours are dealt with in the learning methods and rough set techniques for the context of the line crawling robot.

In the work of Tinbergen [71], he refers to $Ethology$ as a biological study of behaviour. There are four main parts that are included in his discussion,

causation, *survival value*, *evolution*, and *ontogeny* [71]. In the context of behaviours, they have the following meaning. *Causation*, also known as *proximate cause* refers to the event or situation leading up to the behaviour that has occurred. A simple way to think of it is as the cause of whatever the effect was. Next, the *survival value* is the opposite, examining this quantity involves looking at the result or effect of the behaviour that has taken place already and how it has either improved or reduced the survival value of the species in question. The third part is the *evolution* of behaviours and this refers to the how a species has evolved into a certain pattern of behaviours over time based on previous experience or conditions. The fourth and final part referred to in [71] is *ontogeny*. Similar to evolution but on a lifetime scale, *ontogeny* refers to the development of the species over time. These four categories make up parts of an *ethogram* used to study animal species.

Similar concepts from *Ethology* and the *ethogram* are relevant to the work reported here. The reinforcement learning environment used for the tracking tasks parallels the study of behaviours using two of the four parts of *Ethology* that Tinbergen refers to, including *causation* and *survival value*. The cause and effect pair draw parallels with state and reward values

in the context of reinforcement learning. Also, an $ethogram$ is employed and generated throughout the learning process to record and process behaviours. The information regarding states, actions, rewards, selection preference and decisions are all recorded in a table of behaviours. These tables are then processed using rough set theory and approximation spaces to generate a performance metric to adjust the behaviours accordingly for improved performance. The inspiration for these methods stems from the concept of the $ethogram$ and $Ethology$ presented in Tinbergen's work.

3 System Architecture

Building upon the background theory presented in the previous chapter, this section describes the system architecture. First, an introduction to the evolution of the ALiCE II robot design is included, followed by a block diagram representation of how everything fits together and then subsections describing each aspect in the diagram. The design choices are explained along with any alternatives investigated and the associated trade offs.

3.1 The First Prototype

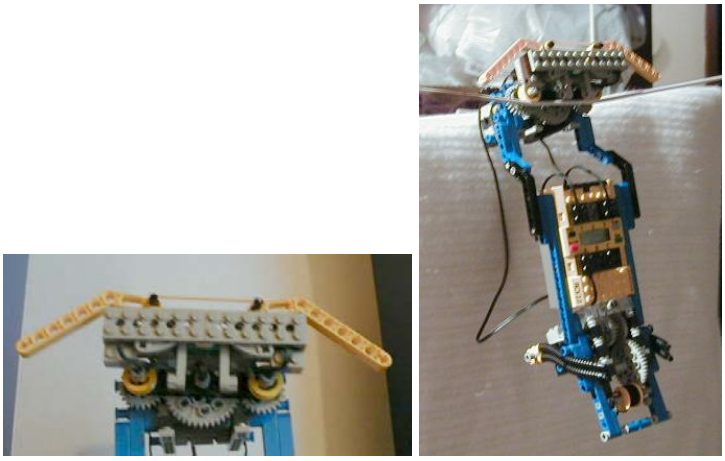
The first step in developing the line crawling robot after the problem had been addressed was to investigate potential prototype designs. This section includes a brief discussion of the design evolution from the conceptual level to the first prototype and proof of concept implementation. The preliminary work used Lego Mindstorms®as a development platform due to the availability and speed of prototyping over a short period of time. Due to a vast number of commercially available sets and many different shaped parts within each, there was no need to have custom machined parts built and the operating system included provided rapid development of simple control

programs, encouraging quick design revisions as needed.

During the first stages of the design, several possibilities were explored in developing a line crawling robot. Existing systems were looked to for inspiration, starting with commercial tram cars and gondolas commonly used as ski lifts or transport for tourist attractions located in hard to reach areas [26]. These were both suitable models of established line crawling devices. A prototype line crawler based on an aerial tram was developed using the Lego®platform (see Fig. 3.1). The first prototype relied on grav-

Figure 3.1: First prototype based on aerial tram

ity and slow speeds of the line crawling robot to keep the wheel securely attached to the line. The single contact point highlighted a potential weak



(a) Improved Line Grip (b) Prototype v2

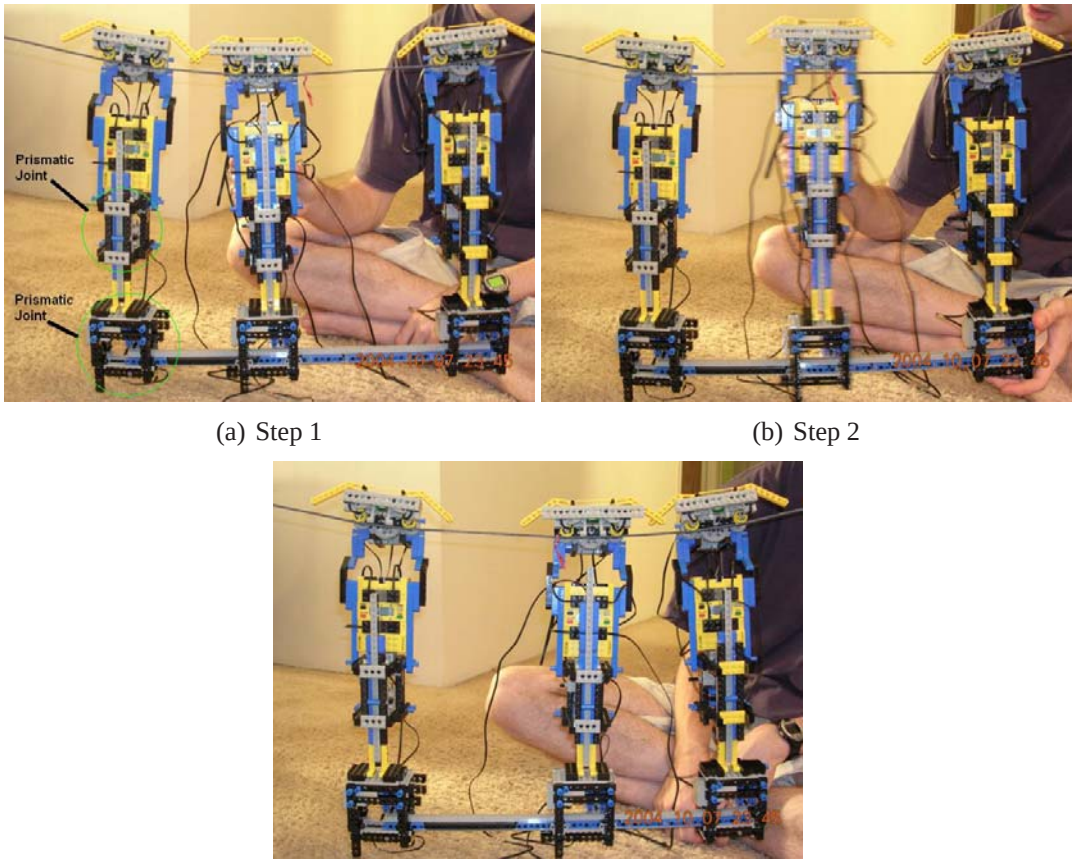
Figure 3.2: Revised version of the prototype line crawler robot

point in the design and warranted a revision to the first generation of the line crawler. The next step was to improve the stability of the design and add more sensory perception so that when obstacles were encountered, rudimentary avoidance occurred. The first set of changes resulted in a new line grip design seen in Fig. 3.2. The new grip included contact sensors facing both directions to avoid collisions and a pair of wheels to distribute contact with the line.

The final revision of the first prototype robot investigated the possibility of taking more evasive action during obstacle avoidance rather than just moving in an opposite direction. To accomplish this, potential methods for disconnecting the grip from the line and then re-attaching were entertained.

Using biological systems as inspiration, $brachiation$ (swinging) motion associated with gibbons was investigated. The main idea was to have a pendulum style of disconnect and reconnect action [1, 57]. Since the prototype materials exhibited too much flex and posed difficulties in spreading the weight evenly, the end result for the design settled on a connected scheme taking advantage of cooperation with multiple line crawlers. Stability from additional support was helpful but it still exhibited weaknesses of its own and required human assistance to perform even the simplest of acrobatics. A simple sequence of figures presents the revised prototype line crawler demonstrating the obstacle avoidance routine in Fig. 3.3

Although the Lego®version of the line crawler was not stable enough to perform the obstacle avoidance routines without external support, the desired concept was demonstrated in the sequence of images for future revisions. Extra motorized prismatic joints were added to increase the degrees of freedom for a wider range of movement. In addition to the standard locomotion of the wheels, both a horizontal and vertical position drive were added to allow movement of the robot when the locomotion drive was disconnected. Both position systems operated with rack and pinion tracks which helped secure the robot during operation.



(a) Step 1 (b) Step 2

(c) Step 3

Figure 3.3: Simple obstacle avoidance with the revised prototype line crawling robot

At this stage in the development of the prototype robot, the capabilities of this platform had been exhausted as the design was already using three processing units, nine motors and six sensors. Coordinating the operation of all of the different aspects of the system was beginning to present a problem for the operating system and controller. Since the ideas were well developed at this point and a number of potential difficulties had been investigated, a solid background was prepared to begin the second version of the line crawling robot design.

3.2 The Second Prototype

This section includes an introduction to the dimensions of the problem, including sample obstacles, and the skywire. After discussing preliminary details, the development of the second version of the ALiCE robot is included in two parts consisting of the line grip and the payload.

The shape and dimensions of the line crawling robot were the first problems addressed when beginning the second generation design. A set of sample distribution towers along Bishop Grandin Boulevard from Pembina Highway to Lagimodiere Boulevard were used to help model the work space available for the line crawling robot. The top of these towers contained a

trapezoid shaped space underneath each line clamp where the skywire travels through. The dimensions of the trapezoid shaped space are shown in Fig. 3.4, giving an idea of the amount of room available for the line crawler to work in. There are two main types of obstacles that were of interest when

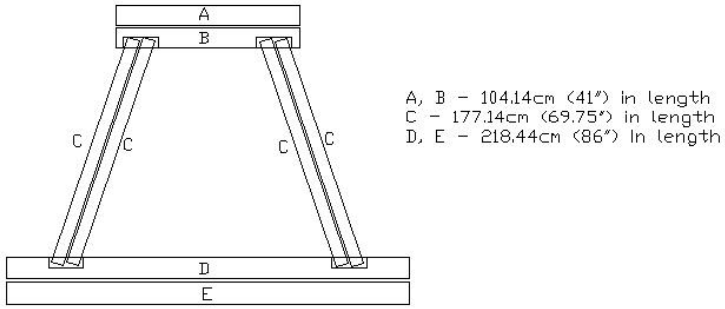


Figure 3.4: Top of tower approximate dimensions

allowing room for the possibility of obstacle avoidance, vibration dampers and line clamps. These two obstacles can be seen in Fig. 2.1, additionally, Fig. 3.5 shows a close up of the same components (samples provided courtesy of Manitoba Hydro). The first prototype had a fixed contact point with the line of travel and needed to be lifted up to disconnect and avoid obstacles, since fighting gravity is already a problem faced by a suspended robot, the new design opted for a grip more like a hand that can open and close over the line. A sample of the skywire was also provided from Manitoba

Figure 3.5: Sample obstacles (2 line clamps and 3 vibration dampers)

Hydro. The diameter of the wire is approximately 9mm and it is twisted presenting an uneven surface. Keeping these constraints highlighted, the next step involved specifying the line grip which is the contact point with the skywire.

3.2.1 The Line Grip

The line grip was the first part addressed in the second generation design of the ALiCE II robot. This was the contact point with the sky wire making it a crucial part of the design since mechanical failure would result in a substantial plummet to the ground and certain damage to the robot. Draw-

ing upon previous experience with the proof-of-concept robot, a new 'C'-shaped clasping mechanism was built, similar to having two fingers above the line and a thumb below. This provided enough contact with the line to secure the grip even in potentially unstable conditions. This section is organized as follows, the wheel design is discussed first, followed by the upper grip and lower grip chassis designs followed by their connections to the backbone structure.

The wheels were designed first, being the direct contact point between the robot and the medium it travels upon. The idea for the shape of the wheels was derived from the proof-of-concept robot. Their primary function was to travel along wire, which calls for a wheel that has a concave profile similar to a pulley wheel or $sheave$ [54]. During the specification process, several extra considerations were made based on materials, sizing and potential pitfalls. A number of different possible materials were considered but the least expensive from a cost and weight perspective ended up being nylon. This material was soft enough to manipulate into the desired shape and rigid enough to support the weight of the line crawling robot. The width of the skywire was known to be 9mm in diameter but with the addition of splices and repairs the maximum possible diameter was estimated

closer to 14mm in diameter. The wheel material was very smooth, providing less than adequate friction to drive the robot up an inclined section of skywire resulting in the need for extra room to improve the traction. Also, the wheel diameter was selected to accommodate a common sized bore as well as providing room to drill a set screw for torque transfer. The shaft size was selected as 1/8” since it was strong enough and a common bore size making it simpler for interfacing to other components in the drive train [32]. The construction drawing for the wheels is provided in Fig. 3.6.

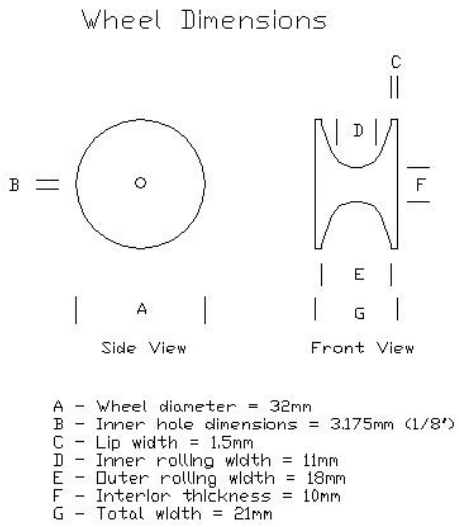


Figure 3.6: Construction drawing for line crawler wheels

The inside of the wheels had sloped side walls based on two diameters of curvature, inner and outer dimensions corresponded to 11mm and 18mm

respectively. The extra spacing for both was provided to add increased traction on the surface that was added after the wheels were machined. The new surface with greater friction consisted of applying an activator followed by a cyanoacrylate-based glue to adhere rubber strips to the nylon wheel. The rubber strips were cut from a bicycle inner tube with a thickness of approximately 1mm and including the height of the glue, the new dimensions after adding traction was 9 and 16mm respectively (see Fig. 3.7. That spacing

Figure 3.7: Nylon wheels with rubber traction

allowed the skywire to rest completely inside the wheel grooves, providing maximum support from the wheels and increasing the integrity of the connection with the sky wire. In addition to fitting a larger wire diameter, the outer measurement of 16mm provided a larger tolerance for connecting to the sky wire, giving a small degree of passive compliance. The thick-

ness of the interior of the wheels is 10mm which provided enough room for the space of the shaft (1/8” thick or 3.175mm), leaving equal spacing of 3.4125mm of nylon on either side of the shaft. The extra space in the wheels next to the shaft allowed room for the introduction of a set screw to assist in transferring torque from the motor drive to the wheels.

The next part of the design consisted of developing the upper chassis. This component of the line crawler housed the wheels and the power train, making it the main interface from the backbone of the line crawler to the skywire. The original shape is much the same as the current revision but several changes occurred throughout the evolution of the design.

Initially, a number of readily available parts were provided and the design was built to accommodate them with some added flexibility in the event of any future changes. After the wheels, the next step of the design was selection of gears. The main purpose of the gears was to translate motion from the DC motor to the wheels. The secondary purpose was to separate the wheels, helping spread out the weight distribution over a greater section of the sky wire and improving stability. In addition, separation of the wheels provided space for the lower section of the grip to mesh from underneath for a relatively simple gear train, using a 1:1 ratio. The torque output from the

locomotion drive was sufficient for direct drive and speed was not an issue since the line crawler was designed for low speed operation. The simplest and most common type of gear is the spur pair seen in Fig. 3.8. For translat-

Figure 3.8: Spur gears used in line crawler power train

ing motor torque directly from the drive shaft to a pair of follower wheels, three spur gears were selected, one connected to the motor drive shaft, and the other two connected to the follower wheels. Both the driver and the follower gears were selected with the same characteristics and ordered from the same manufacturer to ensure proper meshing. With reference to the theory presented in Sec. 2.2.1, the gear specifications are as follows. Since a 1:1 ratio was employed, the driver and follower gears were selected with the same number of teeth. The diameter of the gears were chosen to separate the wheels far enough apart that a third wheel from the lower chassis would

fit in between when both sections of the line grip were clamped onto the skywire. The resulting specifications were a 48-pitch, 65 tooth, 20 degree pressure angle gear with a 1/8” bore. Although there were various values of diametral pitch available, a lower value of teeth-per-inch was selected to provide higher static-torque resistance in the event of any rapid braking action that could potentially cause damage to the gears [50]. The value for the maximum safe tangential load at the pitch diameter calculated using Eq. 2.1 resulted in a value of 460.4 oz-in of static torque resistance, more than three times the torque output from the locomotion drive in the event of a complete stop due to crashing into an obstacle. Although the initial bore size was selected at 1/8”, a later revision of the locomotion drive required an increase to 1/4” bore for the driver gears (see Fig. 3.8, the driver gear is on the left).

Shafting material was selected next for the follower gears and wheels. A bore size of 1/8” was selected when specifying gears as it provided a large enough shaft to easily support the weight of the line crawler. There were a number of shafting sizes available depending upon the application, since it was meant to fit inside the wheels and gears a reduced diameter of 0.1247” (3.17mm) was selected. The smaller dimension was chosen to facilitate integration into the gears and wheels and still work with set screws.

The material selected was solid core precision ground stainless steel, the available alternative was aluminum however the shafts were expected to be supporting a large amount of weight from the line crawler and since they are not large components, the additional weight of using steel over aluminum was negligible and the extra hardness of steel made it an appealing choice since the shafts were expected to experience wear over their lifetime. Spacers and locking collars were added to ensure that the gears were positioned properly and to secure the shafts eliminating any unwanted motion.

With the power train completed, the next step was to build a chassis to house all of the components together. To help minimize the weight of the robot, the chassis was specified using aluminum instead of a steel alloy. Each piece of the upper chassis was specified as 1/16” (1.5875mm) thick aluminum since it was durable enough and still light-weight. The difference in strength between aluminum and stainless steel was insufficient to be a concern and the weight of aluminum components was approximately 1/3 less than steel. The chassis was designed to house the gears, wheels and the dc motor all in one structure. The wheels and gears were separated into individual compartments to make the spacing more manageable and provide a rigid support structure. This was done with separate pieces of aluminum

folded and secured within one another. Since the aluminum was folded, a bend allowance was required to account for excess material taken up in the folding operation. An additional one third thickness of the material was added to the measured lengths of the aluminum inside the fold, and two thirds thickness was included for the exterior measured lengths [73]. This bend model applied to the chassis which was specified for 1100 series aluminum, a common light duty chassis material [73].

An assembly drawing is included in Fig. 3.9 showing a top view of the upper grip chassis including the drive train and the wheels. The final point regarding the upper chassis was fastening the dc motor to the line grip. An unfurled hose clamp was used, drilling it out to fit a machine screw at either end, and then tapping the chassis for the screws to guarantee a secure connection. The dc motor rested on the platform at the back of the upper grip chassis and was secured with the hose clamp. The motor drive shaft connected to the wheels through the gear train, transferring torque and providing locomotion for the robot. At this stage, the upper line grip design was completed and the next step was to include the lower grip chassis.

An important distinction between the upper and lower chassis housing is that the lower part contains only a single wheel and it was a follower, not

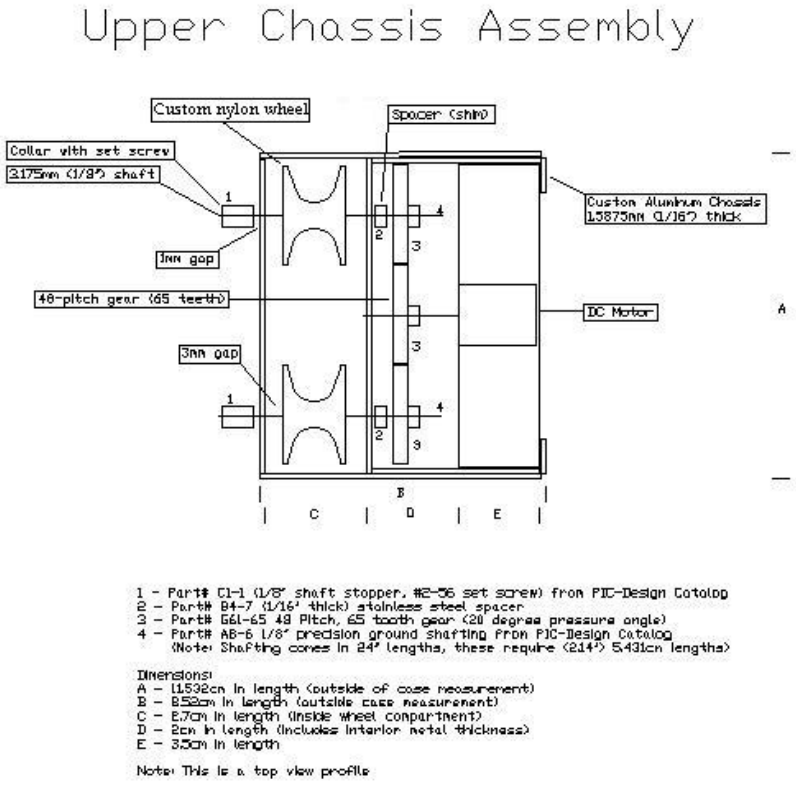


Figure 3.9: Upper grip chassis assembly drawing - top view

on a torque-driven shaft. The main function of the lower line grip design was to improve the integrity of the line crawler's connection to the skywire. Both the wheel dimensions and the available space between the two upper wheels were previously known quantities.

The lower chassis design took advantage of prior knowledge from developing the upper chassis for spacing constraints as they were both fitted for travel on skywire. The lower grip contained a single follower wheel, eliminating the need for a set screw. This introduced another small passive degree of compliance as the extra 3mm of space either side of the wheel allowed for movement when the line grip closes.

The specifications of the lower chassis component used the same material (1100 series aluminum, 1/16" thick) and the corresponding bend model. Extra space was left underneath the wheel in the chassis for attaching fasteners. The lower chassis was then attached to the the backbone completing the grip-like shape and defining a work envelope. The assembly drawing for the lower grip chassis is shown in Fig. 3.10.

After completing the upper and lower chassis components, the next step was to join them together with a backbone structure, completing the line grip. To provide a flexible shape, the connection point with the backbone

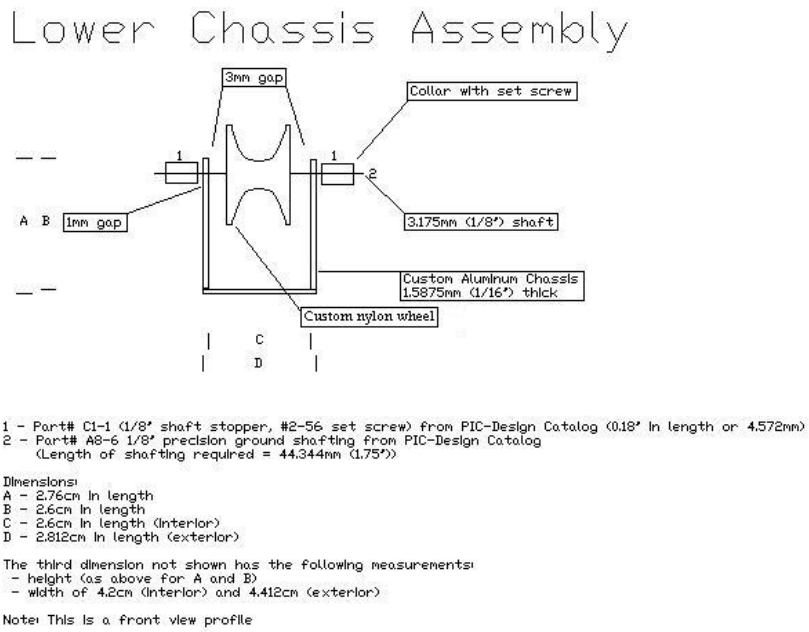


Figure 3.10: Lower grip chassis assembly drawing - front view

for both the upper and lower grips needed to be actuated. Servo motors were selected for the task (see Sec. 3.8).

The servos were intended to support the gripping action, making it necessary to develop a specially formed hinge to attach both the servo and the chassis components to the backbone. This problem required development of a connection that would fasten two perpendicular surfaces to one another.

The dimensions of the hinge for the upper chassis were specified to mount underneath the flat compartment housing the dc motor and the side connected to the servo motor was sized to fit an array of servo horns. The hinge mounting plates were over-sized to allow room for adjustment as needed to secure it. Hole placement for the upper chassis was 2cm farther apart than the motor so the machine screws avoided obstructing the dc motor mount when fastening the hinge. A 90 degree fold to the hinge provided a simple connection to the servo motor. A servo horn was selected that contained pre-drilled pilot holes and was modified by drilling and tapping them for machine screws. With the hinge attaching to the servo, mounting the motor provided some flexibility in placement for aligning the upper grip to provide a nice clean motion for raising and lowering. The hinge was composed from the same 1100 series aluminum as the chassis structures.

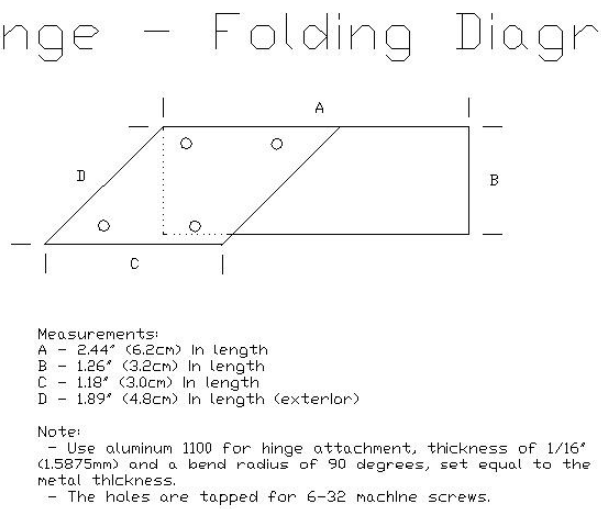


Figure 3.11: Hinge - top view

The dimensions are shown in Fig 3.11 including the location of holes drilled for 6-32 machine screws to fasten with the upper chassis. This size and number of machine screw was chosen to provide a strong enough connection to the upper chassis whilst reducing the required amount of space (0.144” or 3.66mm hole).

The next step in development was to address how to connect the lower chassis assembly to the line grip. Since a similar servo motor was selected to drive the lower half of the line grip, the established hinge design was reused. The only difference with this connection point was that the servo motor location was connected lower on the backbone and as a result, needed

to be compensated for to allow the lower assembly to close under the skywire. Some flexibility was provided in the lower line grip pertaining to variable height adjustment of the lower chassis. The design of the lower grip connection was done in two parts, coincidentally referred to as lower grip attachment part 1 and part 2.

Part 1 is a length of aluminum (1100 series, 1/16” or 1.5875mm thick) that spanned the distance from the hinge to part 2. The connection between the first and the second part included extra tolerance for height adjustment. A bolted connection (#10 bolt) in conjunction with the two individual plates that join part 1 and part 2 provided a total range of movement from 5 cm to 0.5cm above the hinge. The initial design was meant to be manually adjusted. However room for automating (at a later date) was left to allow the line crawler to compensate for changes in skywire widths on the fly. The first part of the lower grip connection is shown in Fig. 3.12.

The second part of the grip attachment was designed with the same shape as part 1 on one side and the same dimensions as the base of the lower chassis on the other. The resulting component completed the lower grip, linking it into one piece. The structural drawing of the second part can is shown in Fig. 3.13. The built-in manual adjustment was secured with

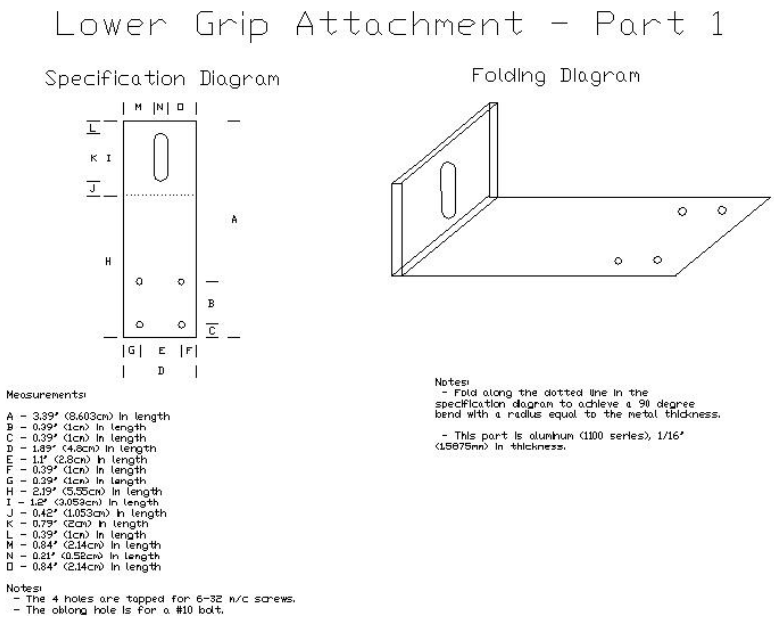


Figure 3.12: Lower grip connection - Part one

a wing nut and lock washer combination. The adjustable position proved useful when servo motors were replaced as there was enough room to alter the height for the new servo and still maintain connection to the skywire.

The final part of the line grip design was a backbone structure to join the upper and lower grip into a complete unit. The original design specified the same 1100 series aluminum material which was sufficient to support the initial weight. As the design evolved and extra components were added the weight of the grip exceeded the strength of the backbone. A revision to the design substituted the original part with a piece of square aluminum tubing

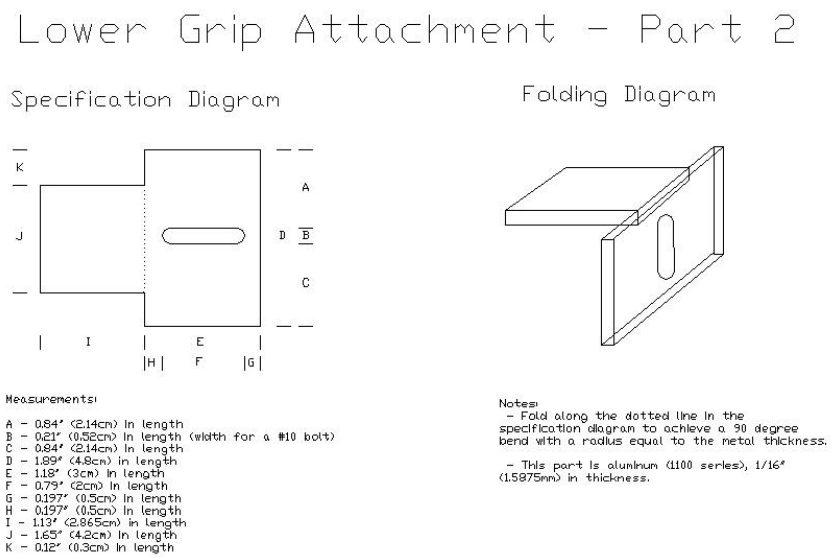


Figure 3.13: Lower grip connection - Part two

drilled out with the same pattern to support the upper and lower line grip (see Fig. 3.14). The servo motors were connected directly to the backbone and secured using unfurled hose clamps due to the strength and malleability of the metal. Holes were drilled in the clamps and the backbone was drilled and tapped to ensure a rigid connection. Once screwed down, the servo motors were able to repeatedly raise and lower their respective grip mechanisms accurately. The completed line grip construction drawing can be seen in Fig. 3.15, showing how all of the components fit together. This concludes the discussion of the detailed design for the line grip. The next part of the design included developing and connecting a platform to support a payload

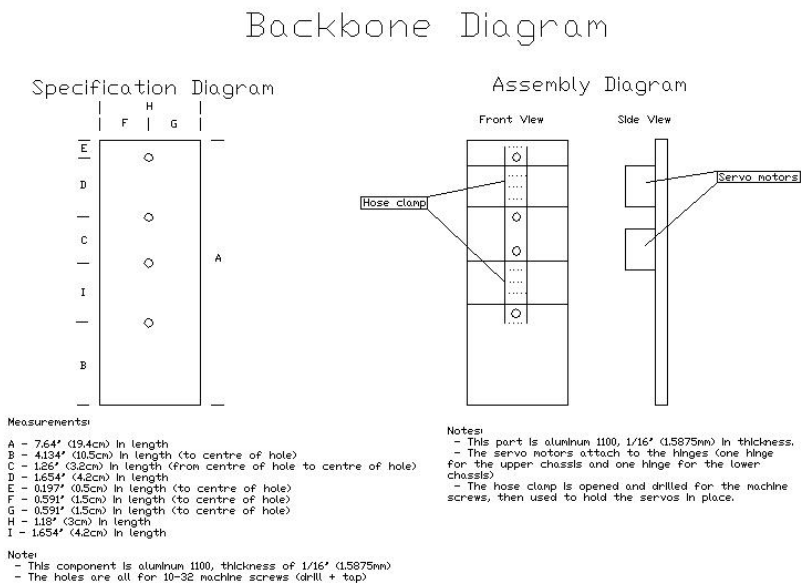


Figure 3.14: Line grip - backbone

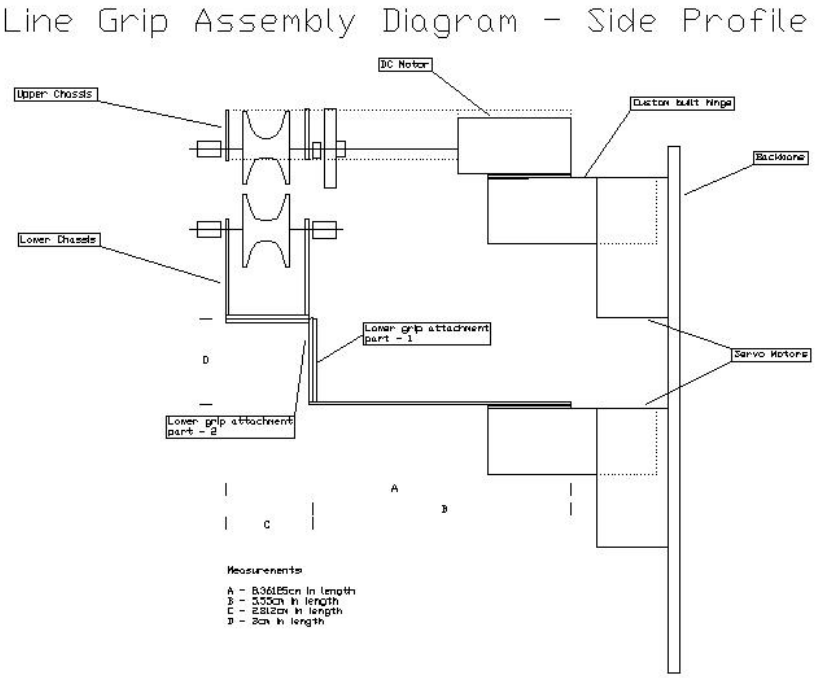


Figure 3.15: Complete line grip construction drawing

for the line crawling robot.

3.2.2 Adding a Payload

The line grip design was intended to support the later addition of a platform for supporting a payload. Once the decisions were made for necessary equipment to promote basic survival of the line crawling robot, the platform design began. An important principle behind the design decisions was to keep the weight and space to a minimum. This section includes a discussion of the platform design and how it attached to the line grip.

To keep things small and straight forward to manufacture, a rectangular base was selected with added space for future expansion of the payload. The main control board dictated the size and shape of the platform with some additional space left for power supplies, control peripherals and room for expanded payloads. The result was a rectangular platform constructed out of the same 1100 series aluminum as the rest of the robot. The dimensions were 29.4cm in length x 15.2cm in width, with standoffs provided to raise the control board allowing the power supply to fit underneath. During preliminary verification, the weight of the line crawler was discovered to be too heavy. The platform material was revised, switching from aluminum to

acrylic of the same thickness. The main concern in switching to acrylic was rigidity and support for the payload. However, after performing strength testing experiments, the new structure held up and ended up being the chosen material for the platform design, providing a weight reduction of 70% compared to its metal counterpart. Construction of the acrylic platform was done by hand, cutting components from sheets of material and then using cyanoacrylate-based glue to bond everything together for a permanent fit.

To connect the platform and the line grip, a handle and mounting bracket were developed. The handle for the platform was centred across the middle and contained evenly spaced holes tapped and drilled for 6-32 machine screws. The pattern of the holes matched with the mounting bracket design allowing for placement of the line grip anywhere along the handle to help balance the payload with positioning. The other part of the mounting bracket was drilled and tapped to match the hole pattern on the backbone, fastening with 6-32 machine screws as well. The completed line crawling robot can be seen in Fig. 3.16. This includes the acrylic platform with the revised backbone material, the entire payload, all of the wiring and the sensor array. The last point for discussion of the second prototype robot covers the work envelope of the line grip relating to its range of motion.

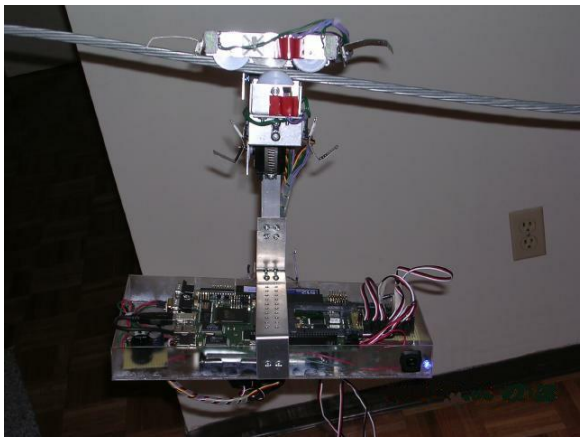


Figure 3.16: Line crawling robot - Second generation (ALiCE II)

3.2.3 The Work Envelope

This section covers the details for the space available that the line grip required based on dimensions of obstacles expected to be encountered. The sample obstacles provided by Manitoba Hydro demonstrated that a variety of shapes and sizes existed for each part. The largest samples were chosen for the design providing maximum spacing when planning the work envelope. Obstacle diagrams were developed to study the space taken up by vibration dampers and line clamps. Although this section appears after the line crawler design in this report, it was covered before and the information gathered was used to identify space constraints prior to development of the robot.

The first obstacle considered was the vibration damper. Figure 3.5 shows that the shape of the dampers are non-uniform from one end to the other. To simplify matters the largest dimension was used to create a rectangular shape around the entire obstacle providing a worst case estimate. The obstacle diagram for the vibration damper can be seen in Fig. 3.17, giving a guideline for the maximum area that is taken up by the obstacle. In addition

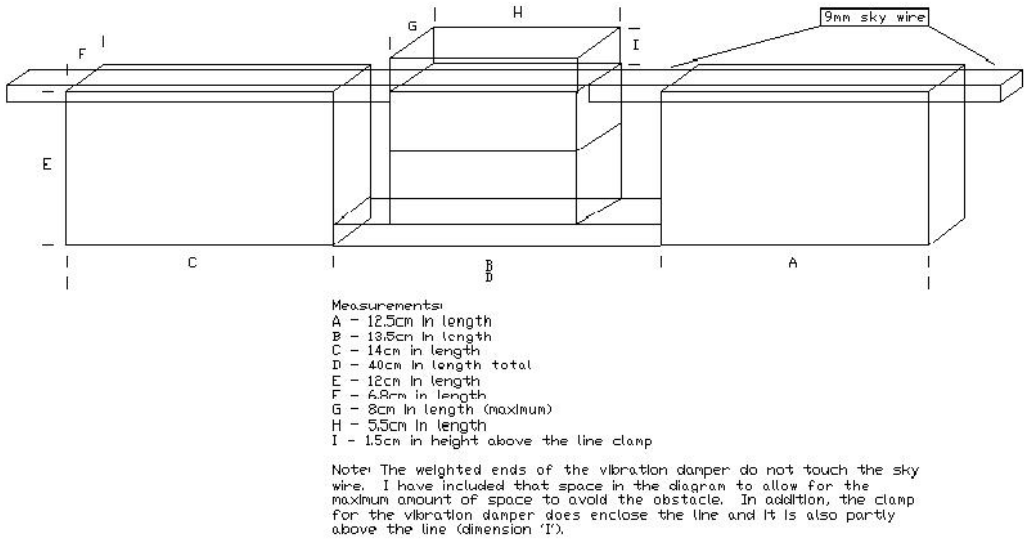


Figure 3.17: Diagram of vibration damper obstacle space

to the space taken up below the skywire, the dampers take up space above the sky-wire (approximately 1.5cm). In order to avoid the vibration damper, the line grip needed to move both the upper and lower parts of the grip.

The amount of movement for the two differed greatly as the upper grip only needed to move 1.5cm before being able to roll along the top of the obstacle . Conversely, the lower grip had to move completely out of the way to the fully open position, pointing straight down. The space taken up by the largest damper was 3.4cm either side of the skywire and up to 12cm below the skywire. The line grip work envelope was able to position the lower grip chassis at 17cm below the skywire, moving the entire lower arm 7cm to one side, giving ample room for the future development of an obstacle avoidance routine.

The other obstacle diagram for the line clamp was much simpler to develop as it can be represented by a single 3D rectangular shape that extends below and above the skywire. The line clamp was considered to extend above the skywire as it connected directly to the tower structures fastening the line. The depth of the object (referenced as 'C' in Fig. 3.18) corresponds to the maximum possible depth that the bolts hang below the line as they were the lowest point. The space required for the line clamp was 3.5cm either side of the skywire and 5.5cm below, fitting easily into the flexibility of the lower line grip work envelope previously discussed. The upper line grip opened 90 degrees in the clockwise direction, providing 6.5cm of space

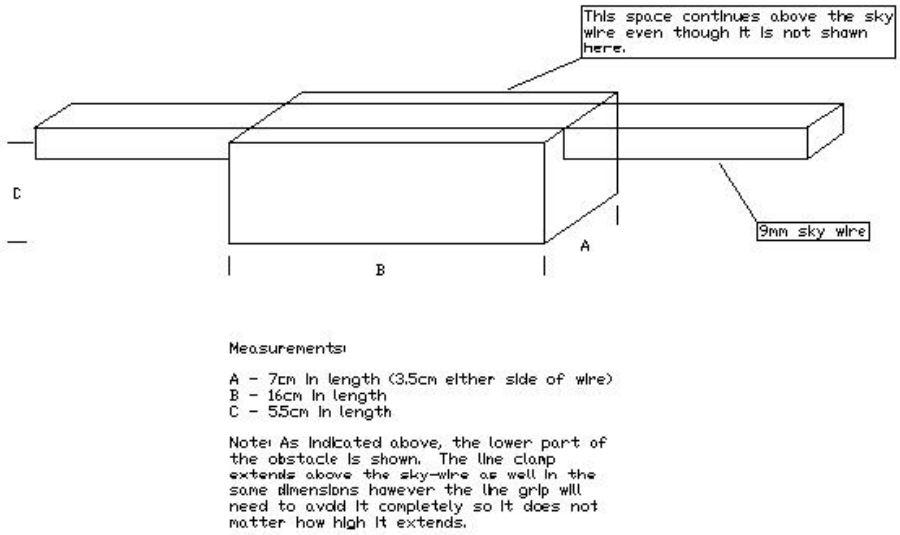


Figure 3.18: Diagram of line clamp obstacle space

between the skywire and the gears which were the lowest protruding component of the upper line grip, also providing ample space to avoid the line clamp with both the upper and lower grip movement.

After examining both types of obstacles and establishing the space constraints for each, the next step was to outline the work envelope of the line grip to ensure it was capable of avoiding them. When assessing the work envelope, all of the protrusions above and below the chassis assemblies were considered to gain a better picture of exactly how much space there was between grip and obstacle at any position of the line grip. The fully closed grip and all of the corresponding measurements are seen in Fig. 3.19. The wheels and gears protruded above and below the chassis assemblies and the position servos moved 90 degrees in opposite directions (the upper grip servo rotated clockwise, whilst the lower grip servo rotated counter-clockwise) providing a picture of the available range of movement, leaving room for an obstacle avoidance plan.

3.3 The System Diagram

The next part of the system architecture discussed are the components that make up the systems that drive the robot. A block diagram showing the

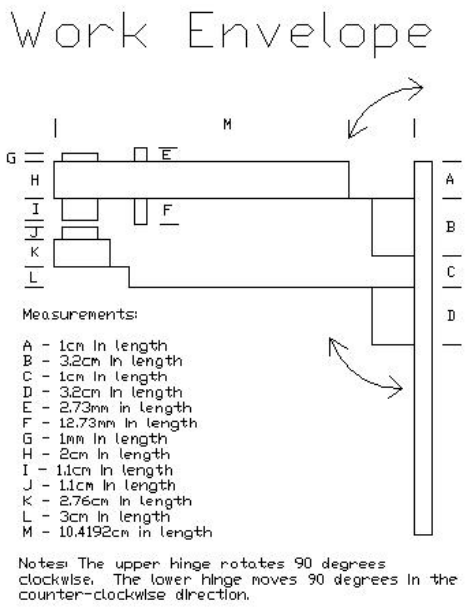


Figure 3.19: The work envelope for the line grip

various components and how they interconnect is provided in Fig. 3.20. As seen in the connection arrows, not all components are bi-directional in nature. The following sections of this chapter elaborate further on the details of each component.

3.4 The Vision System

After developing the structure of the robot, the next step was to provide a means for acquiring images. A number of different options were considered and the final selection was based on system constraints, required specifications, ease of interfacing, and cost.

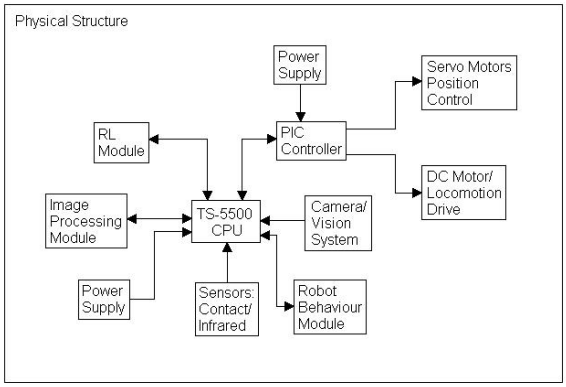


Figure 3.20: System level block diagram with interconnects

Digital photography equipment had become readily available and inexpensive at the time of construction so a number of options were investigated. One of the primary considerations was to keep the camera features to a minimum as the line crawling robot only required still imagery. The nature of an autonomous design also warranted a minimum amount of power expenditure, weight and simplicity of interface. Surveillance and spy camera technology was considered due to the nature of the size, however the cost and power requirements were surprisingly prohibitive. Web cameras provided a reasonable alternative for acquiring images. These types of cameras were ready-made to interface with computers and basic models provided few features beyond taking pictures, minimizing the space and power requirements of the device. A single camera (monocular vision) system was

selected. This still provided the necessary imagery and at the same time reduced the cost, power requirements and weight of an individual robot.

Although there were many options available, the field of choice was narrowed to two possibilities. Both Creative webcam products, the NX and the NX Ultra. The cameras were similar in size but differed in shape, with the NX being more rounded and bulky (see Fig. 3.21) where the NX Ultra was more narrow and slender (see Fig. 3.23) complete with flat panels, easier for mounting. Both cameras supported USB interfaces, making them easy to connect directly to a processing board eliminating the need for separate power inputs. The maximum resolutions offered by each camera differed with the NX rated at 640x480 and the NX Ultra reaching 1280x960 pixels. Although preliminary resolutions used during the experimental phase were low, the appeal of the NX Ultra for future expansion, space constraints and ease of mounting resulted in it being an easy choice for the vision system.

After selecting the camera, the next problem to tackle was how and where to mount it on the acrylic platform. The Creative NX Ultra comes complete with a stand built in ready for perching on a monitor or desk. For the purpose of mounting on the line crawling robot the stand was removed, leaving the body of the camera as the direct contact point. Based on load balancing and

Figure 3.21: Creative NX camera

to provide the best possible views for the camera, it was mounted so that the neutral position was in the centre of the platform underneath the robot. This location provided adequate protection for the camera and left enough room that it did not interfere with the cabling during movements. The range of motion (work envelope) associated with the camera was limited by the wire connections and the maximum rotation of the position control motors. A diagram demonstrating the camera's field of view can be seen in Fig. 3.22. Although the camera was positioned for forward travel of the line crawling robot, it was mounted with an adhesive foam tape, making it possible to remove and remount in alternate orientations as needed.

The last point of this discussion includes a look at how the camera was mounted to the position control servos allowing for the field of view shown

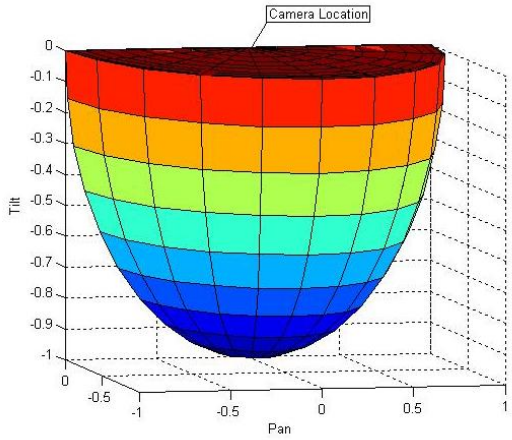


Figure 3.22: 3D field of view for the camera

in Fig. 3.22. Rather than any additional mounting hardware, the camera was connected directly to a position control servo which was in turn, mounted directly to another position control servo that was adhered to the underside of the platform (See Fig. 3.23). All of the connections were made using indoor/outdoor foam tape with a maximum strength rating guaranteed to secure the camera and servos (rated to 907g). This provided two degrees of freedom when operating the camera, pan and tilt. The constraints of movement for the camera stemmed from the range of the servos (on average 180 degrees), the wired connections from the camera and servo motors, and finally from the platform which restricted any upward movement of the camera. Although the field of view is approximately equal to one quarter of

Figure 3.23: Creative NX Ultra mounted to position control servos connected to the robotic platform

a sphere, this was sufficient to view details on the tower structures as the line crawling robot rode along the skywire at the top of towers looking down on everything. Positioning the panning servo so that it looks in the forward direction of travel at 0 degrees and then moves +/- 90 degrees from the origin allows for a complete field of view in the direction of travel. The alternative would have been to position the origin perpendicular to the direction of travel to provide a field of view on one side of the line of travel. Either option was viable as the intention of the line crawling robot design was to extend to multiple robots (see Fig. 3.24) on the skywire working in cooperation. Along with multiple line crawlers, having one line crawler viewing the left side of travel and another viewing the right side would provide a full

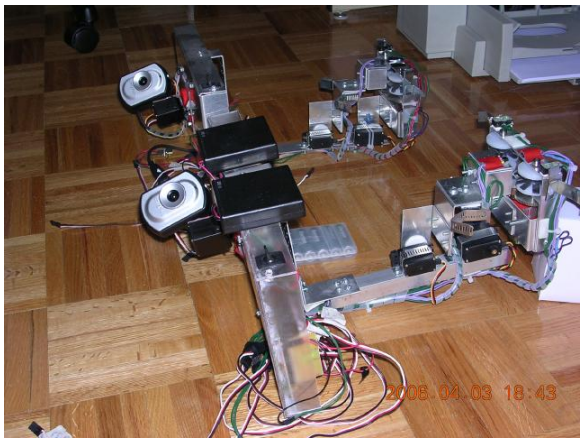


Figure 3.24: A pair of line crawling robots with mounted cameras

field of view below the line crawling robots.

3.4.1 Image Processing

A description of onboard image processing for the line crawling robot is included in this section. At the time of writing this report, the template matching approach was the main method employed (see Sec. 2.5.1). However, work was being done to compare the efficiency and performance of the average grey level tracking method. A description of the template matching method details are provided here along with some insight into the average grey level method and how they differ.

The images provided for processing were all gathered using the Creative NX Ultra camera via the SPCA50X USB Linux driver [79]. The steps

involved in the target tracking task for image processing were as follows. First, a target was acquired, this was done manually during the experimental phase by positioning the camera toward a target and grabbing a still image used for the template. Due to the restrictions of the computational power of the TS-5500 controller, the minimum image size of 160x120 from the camera was decimated by a factor of four, yielding an image for processing of 40x30 in size and using greyscale pixel values. Even though the image being processed was reduced in both size and colour, it was still possible to discern the sample target (see Fig. 3.25). The template size for the reduced image was 28x21, leaving a possible space for movement away from the centre at 12x9 pixels. The image space was divided up into 9 states (discussed further in Sec. 3.11.1) and a corresponding set of actions were available for each state to provide a means to move so that the target remained centred in the camera field of view.

The average grey level method of tracking was similar with respect to the number of states, actions and rewards, but locating the target was handled differently. Rather than matching a template, prior knowledge of the size of the images were used to divide them up into subsections and then discover average grey levels contained within. The targets used during experimental

development contrasted well with their backgrounds, generally employing a dark target on a light background (see Fig. 3.25). The size of the subsections

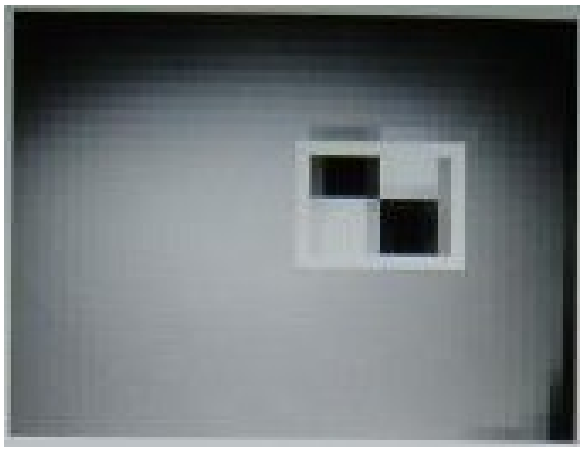


Figure 3.25: Sample target, decimated and converted to greyscale

of any image were dependent upon the original dimensions. For the experimental work provided in this report, the size of the input image for both tracking methods was kept uniform at 40x30 pixels. To divide up the 40x30 image, a total of 15 cells were used in a 5x3 configuration providing the potential for more states than the 9 included in the template matching method. There was also a reduction in accuracy dependent upon the resolution of the cells as the centre of a cell with the lowest grey level was considered the target instead of the centre of a template placed on any pixel. The maximum resulting error was +/- 0.5xM in the horizontal direction where M was the number of rows in a subsection of the image and +/-0.5xN in the vertical

direction, where N corresponded to the number of columns in a subsection of the image.

The average grey level method was configured to assume that a dark target would be present on a light background. This required some prior knowledge about the background and limited the technique as opposed to template matching which was able to operate regardless of what the background was like. A possible extension of this method could employ variance instead of average grey levels using the same structure. This may be more well suited for the problem of spotting targets like smooth surfaced insulators with a background consisting of fields, trees, towers, or anything with potentially higher variance. The average grey level method was implemented for preliminary experimental work but it was not expected to perform as well as template matching in the experimental environment.

3.5 The TS-5500 Computer

During the development stages of the line crawling robot, work was being done in the computational intelligence lab (CILab) using a single board computer. Taking advantage of the work and experience of the CILab group, selecting the TS-5500 (Technologic Systems) as the main processing board

for the ALiCE II robot was a simple decision. Although there were a number of other options available for controllers, the availability, cost and prior knowledge made the TS-5500 the clear choice.

The TS-5500 was a full-featured single board computer chosen to coordinate the various systems of the line crawling robot. It was modelled after the x86 family of PCs and as a benefit had the same sort of memory map, simplifying access to system I/O [69]. The central processing unit onboard the TS-5500 was the AMD Elan520 processor, an 0x586 class processor operating at 133MHz [12]. Input ports included two universal serial bus (USB) ports, a PCMCIA (personal computer memory card international association) slot, three communication ports and an RJ-45 network port were available, providing for a number of possible interfacing solutions. The digital I/O and the A/D converter provided support for low level devices and sensors. As shown in Fig. 3.26, the TS-5500 is a compact unit. The dimensions are 17.8cm in length and 13.2cm in total width. When a card was plugged into the PCMCIA slot, an additional 5.2cm was added to the overall length, still leaving enough room in the line crawler platform for its payload and room to spare.

The TS-5500 also provided good support for software and software-

Figure 3.26: The TS-5500 single board PC

related processing. An onboard flash drive of 2MB was present, as well as 32MB SDRAM (synchronous dynamic random access memory), and a compact flash port for additional memory. A minimized version of the Red-Hat Linux 9 kernel (Shrike) operating system [29] was installed and used to manage the line crawling robot. Using a large compact flash card with 512MB of storage in the expansion port provided ample support for the operating system and program code for running and interfacing with all of the robot systems.

The array of features provided by the TS-5500 for interfacing and computational power made it very appealing but there were a couple of drawbacks as well. First, the power requirements were more demanding than initially expected. Quiescent power draw of the TS-5500 was measured at

approximately 0.5 amps. During heavier processing stages such as control of devices and communication, the current draw increased to approximately 1 amp. The power requirements for this type of board warranted some design changes in the power supply (see Sect. 3.9). The other drawback came in the form of control of the digital I/O and timing. To control position servos, generation of fine resolution control signals was required (in the order of microseconds). The TS-5500 was unable to provide such a fine resolution using its onboard real-time-clock chip which provided at best 122 microsecond square waves. Using a software interface to the main processor was no better since it was only accurate to the millisecond range. Rather than switching to an alternate platform, a separate PIC controller was added to interface with the motors (see Sect. 3.6).

Although there were a couple of drawbacks associated with the TS-5500, overall it provided a solid performance as the brains behind the line crawling robot. Having a full computer on a single board allowed interfacing with many types of devices. Using the Linux environment, GNU C was used for code development providing a familiar and powerful interface for programming. The benefits associated with the TS-5500 outweighed the drawbacks making it an easy choice to fit into the system architecture as the

main controller for the robot.

3.6 The PIC Controller

Since the TS-5500 was unable to manage fine resolution clock operations for controlling servo motors, a separate controller was added. The total number of motors onboard the line crawling robot was five. There were four position control servos and one dc motor for the locomotion drive. Rather than split up control of the two types of motors, a control board was designed to operate all five of them. A PICMicro controller was used to operate all of the motors. This section includes a discussion of selection for the device, as well as what was needed to complement the PIC for interfacing, power requirements, and finally layout and creation of the control board.

When looking for a secondary controller to operate the motors there were many options available ranging in price, size and power requirements. During the original building phase of the line grip, a Handy Board [31] was used to operate all of the motors and sensors for verification purposes. The Handy Board was an all-in-one controller providing its own proprietary high level language for programming (Interactive C), as well as hardware interfaces

for controlling servo motors and dc motors and for running both analog and digital sensors [31]. The Handy Board was dropped since the TS-5500 eclipsed its functionality on all levels except for fine control of clock signals and since it was large and required more power, smaller controllers were favoured in its place. When considering small-scale controllers, there were many available, including products from Motorola, Zilog, Texas Instruments, and Microchip to name a few. For the most part, each of the controllers provided similar features, with the odd difference between them. As a result, any of them would have been sufficient. However, there were several reasons why the PICmicro was selected over the others. First, some members in the CILab (including myself) had previous experience with PICs and the technical staff on campus provided support and lab facilities that were already in place. Also, they were one of the least expensive alternatives at around five dollars per unit. In addition to that, development tools such as MPLab and CC5x were readily available online and free, both excellent products, making testing and implementing code for the PIC much faster. There was also a wealth of information documented online from many sources due to the popularity of PIC controllers (for a few examples, see [77, 62, 53]).

Due to a wide range of PIC processors available, a list of needs was the starting point for selecting the device. Requirements for the controller included the ability to provide fine resolution clock pulses for servo motor control, support for communication in the form of an onboard UART (Universal Asynchronous Receiver Transmitter), support for a minimum of five digital I/O ports, and flash memory for programming and allowing easy updates of new code revisions. With those needs in mind, a mid-range device in the 16 series was chosen, the 16F871 [49]. This PIC met all of the requirements with room for expansion, offering 33 I/O channels, an A/D converter, 2K of memory for instructions, an internal or external clock and an onboard USART (Universal Synchronous/Asynchronour Receiver Transmitter) for interfacing seamlessly with the TS-5500. Running off a 5 volt power supply, current draw of the PIC was rated typically at 1.6mA during operation at a clock speed of 4MHz and less than 1μA in standby mode adding very little extra power demand on the robot [49]. The most important condition that needed to be satisfied was fine resolution control of the digital outputs to create position signals for the servo motors. The 16F871, running at 4MHz was able to provide a resolution of 10μS which was inside the desired range of 2-12μS that was sought after. In order to achieve a fine resolution with

consistency, an external crystal oscillator circuit was necessary as the internal RC-oscillator on the PIC did not have the same quality factor or stability. A 4.000000MHz crystal was selected with 15pF stability capacitors for an accurate and stable clock speed [36]. The next problem explored was how to interface the PIC to all of the different input and output devices.

The PIC was unable to interface on its own with all of the external devices. Difficulties were encountered with communication using the RS232 standard (see Sect. 2.2.3) and with operating the dc motor. To solve these problems, additional components were added to the control board in the form of an RS232 transceiver and an intelligent h-bridge.

The reason that the PIC required an RS232 transceiver is because the standard voltage levels for PIC outputs corresponded to CMOS logic levels, low between 0-1.5 volts or high between 3.5 - 5 volts [30] and as discussed in Sect. 2.2.3 that did not coincide with bipolar logic (see Fig. 2.6). Also, the voltage range for RS-232 communication is from +/- 3 to 25 volts, low voltages can be subject to distortion error due to attenuation in transmission cables. These problem were addressed through the addition of a Linear Technology RS232 transceiver (part#LTC1383) [28]. The transceiver operated from a five volt supply and through the use of internal circuitry

and some external capacitors it boosted the output voltage to a +/- 7 volt swing [28], easily interfacing with the RS232 protocol. Since this was a low power device, rated at 220μA unloaded and it operated off a single five volt power supply, it was a suitable choice to meet line crawler's needs.

Control of a dc motor required the addition of an intelligent h-bridge to the control board. Without getting into the theory of operation in too much depth, the main idea of an h-bridge is to allow control of a dc motor with logic level voltages and the power to the motor can be much higher depending upon the size of the motor. An h-bridge is designed with four transistors around a motor and these four create two separate paths from the power supply to ground through the windings of the motor when connected [33]. This allows control of the motor in either a forward or reverse direction. A simple example can be seen in Fig. 3.27 where a partial circuit has been drawn for convenience. Through the use of the h-bridge, logic level inputs to I1, I2, I3 and I4 turn on the transistors which provide a path for the current from the voltage supply V+ to ground. The power supply V+ can be any dc value and is commonly higher than the levels driving the transistors. For the case of the line crawling robot, the input control voltages are derived from the five volt power supply whilst the dc motor voltage is 9.6 volts. To

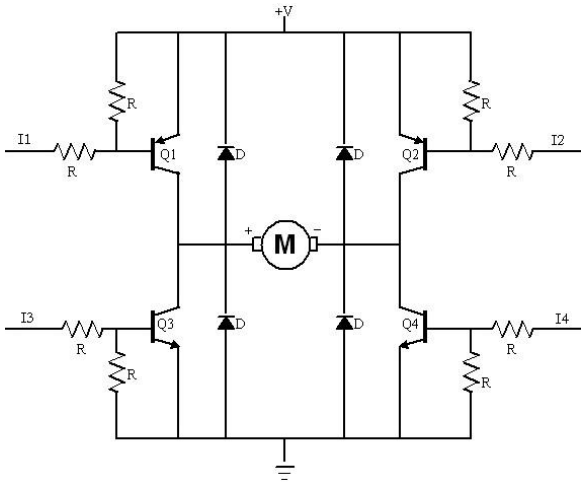


Figure 3.27: H-bridge example circuit

clarify the forward and reverse operation of the motor, a quick scenario is included. When the robot received a cue from the TS-5500 to drive forward, both I1 and I4 are triggered, turning on Q1 and Q4, generating a path for the current through the motor in the positive direction in turn causing it to drive forward. To reverse, the opposite inputs (I2 and I3) are driven high and the current path goes through the motor in the opposite direction. When stopping, there were two possibilities depending on the nature of the stop required. The first possibility was a slow stop where power to all inputs was removed and the motor coasts to a stop. The second possible stopping method is braking, achieved by turning on inputs I1 and I2, which in turn forces the motor to stop turning by having it work against itself. The rea-

son this works is because a motor generates a voltage (like a generator) and when the terminals are both connected to the same point (either high or low voltage), it is seen as a short across the terminals and the generator voltage will counteract itself as it will seem that an equal and opposite voltage is being applied across the terminals, causing a braking effect [7].

When examining the possibilities for driving the dc motor, a few alternatives were investigated. First, the possibility of building a standalone h-bridge much like the circuit in Fig. 3.27 was examined and then discarded shortly afterwards due to space and power requirements. Several h-bridges contained on a chip complete with intelligent control were found. Versions from Allegro, National Semiconductor (NS) and Texas Instruments (TI) were compared. The device with the simplest interface and lowest power requirements was manufactured by TI. The TPIC0108B [70] from TI was selected to operate the dc motor locomotion drive for the line crawler. This chip was a surface mount package intended for low power or battery operated circuits so special considerations were taken into account when including this device in the control board layout.

The input voltage requirements for the PIC and its associated devices were similar. The RS232 level transceiver operated from five volts, as did

the PIC, however the TI chip had a rated minimum input voltage of six volts. In order to run multiple power levels to the different devices, two taps were made from the battery pack for the control board. The first tap was unregulated dc straight from the batteries at 9.6 volts. This was the supply voltage for the motors as well as the h-bridge chip. A second tap was after a voltage regulator (see Sect. 3.9), providing a regulated 5 volts to the PIC and the transceiver. The benefit of taking two taps off the same battery pack was a reduction in circuitry to provide two voltage levels and the h-bridge chip and motors were resilient enough that any fluctuation of input voltage would not affect normal operation until the batteries started failing. A completed schematic of the interconnects for the PIC control board (see Fig. 3.28) was used as an aid to set up the board layout.

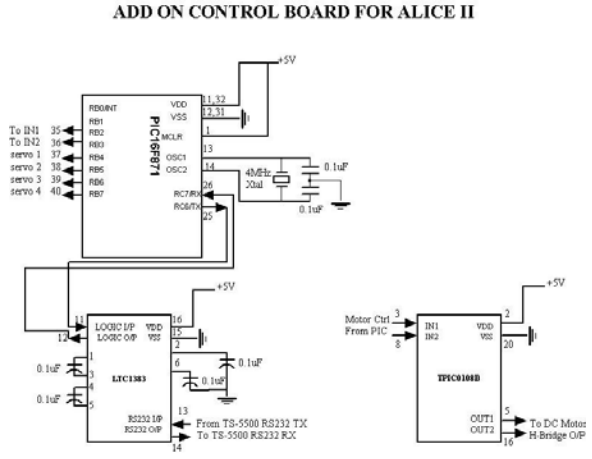


Figure 3.28: Control board schematic

The last stage of the PIC control board development was layout and construction of the prototype. When allotting space on the control board for each device, the size, the number of discrete components it required and the position relative to the other devices was included in the layout design. The largest integrated circuit (IC) was the PIC so it was placed first, followed by the RS232 transceiver and finally the motor control chip. Since the h-bridge IC was a surface mount package special considerations were necessary to add it to the perf-board construction. A surf-board was added with a single inline set of pins to allow for easy insertion into a through-hole board. Each of these devices had some additional through-hole hardware to accompany them. The PIC was operated with an external crystal oscillating at 4MHz, this required a crystal and a pair of capacitors. The RS232 transceiver required an additional 4 capacitors to operate its charge pump circuit for boosting voltage levels and finally the h-bridge control IC had an additional coupling capacitor for the power supply inputs for safety. Adding each of these devices to a compact board arrived at dimensions of 8.5cm x 6.5cm, shown in Fig. 3.29. Each of the through-hole IC's and the surf-board were socketed to allow for quick dis-assembly as needed and safer soldering avoiding heating up any of the electronics by removing them. In addition to

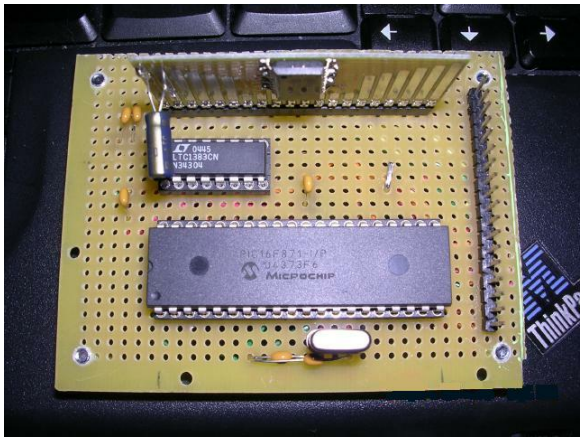


Figure 3.29: Line crawler robot motor control board

that, socketing the PIC allowed for easy access to remove and install it during times when the operational software was updated and the device needed to be reprogrammed. All of the wiring and soldering was done on the underside of the perf-board requiring stand-offs, providing space underneath the board to avoid damage once installed in the platform.

With the addition of the PIC control board, low level control of devices external to the TS-5500 were made possible with a minimum additional cost, both financial and power related. Running at a clock speed of 4MHz, the PIC was able to generate fine resolution signals for position control of the servo motors as well as operating the dc motor and there were also additional ports left available for future expansion. The interface between the PIC and the TS-5500 was accomplished with a serial link and a hardware

UART at each end. The transmission protocol for communicating was developed for speed and efficiency and has a more detailed discussion in the next section.

3.7 Communication Protocol Between the TS-5500 and the PIC

Once the two control boards had been established, a means to communicate between the two of them needed to be devised. This section covers the choices made and the reasons behind how the communication protocol was developed. A general discussion of an overview of the protocol is included, followed by a break-down of how devices were selected and commands were issued.

During the construction stage for the PIC controller a number of tests were done to verify proper operation and communication with the device. Several considerations for the communication protocol arose from the results and problems encountered during testing. To keep the demand on the PIC to a minimum, one byte of data was selected to contain all of the information necessary to issue a command. The PIC USART module contained a 2 byte FIFO buffer (First In First Out), allowing up to two bytes of information to be stored at any given time without incurring a loss of data [49].

Communication was originally designed to be uni-directional with the TS-5500 issuing and the PIC executing commands. At a later stage in the evolution of the line crawling robot, two-way communication was included allowing feedback from the PIC. Transmission of information in both directions used the serial port on both controllers in asynchronous mode using an 8-N-1 protocol (signifying 8 bit transmission, no parity bit and one stop bit). A communication speed of 9600 bits per second was selected as it provided the best ratio of transmission speed versus bit error rate [49]. Each command byte was separated into two parts, the lower three bits represented the device selection with two exceptions for the value of six and seven which correspond to a query and reset command respectively. The upper five bits corresponded to the command issued to the device (see Fig. 3.30). With

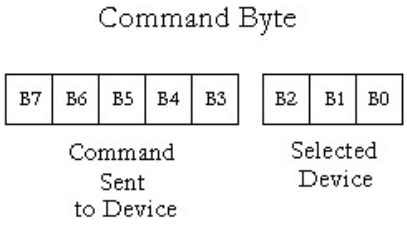


Figure 3.30: Command byte separated into device selection and command bits

three device selection bits there was a possibility of addressing eight sepa-

Bin.	Dec.	Description
000	0	DC motor is selected
001	1	Camera panning servo is selected
010	2	Camera tilting servo is selected
011	3	Upper chassis grip servo is selected
100	4	Lower chassis grip servo is selected
101	5	—Currently unused—
110	6	Position feedback command for camera servos
111	7	Reset camera servos to origin positions

Table 12: Device selection using lower 3 bits of command byte

rate devices. The five command bits allowed for thirty-two possible commands available for each device.

Device selection was needed to choose which motor to control and since there were only five choices, that left room for three additional commands. As the system evolved, it was discovered that a reset and a query command were useful for operating the camera servos, taking up two of the three extra spots, leaving one more for future expansion. The command tables presented in this section link bit patterns to commands. First, the device selection bits can be seen in Table 12 The device selection bits are self-explanatory with the exception of the sixth and seventh commands. The sixth (or 110) command is received as 'XXXXX110' where the X's are any value. This command had the PIC send feedback to the TS-5500 containing positions of the camera servos so that the TS-5500 could quickly locate

where the camera was pointing. The position of the two camera servos were reported using two bytes, the first byte was the pan servo and the second byte corresponded to the tilt servo. Both position bytes ranged in value from 0 to the number of possible steps for the brand of servo in question. The reset command (or 111) is received similarly as 'XXXXX111' where the high bits can be any value. Upon receiving a reset, both camera position servos were reset to their origins. For a number of operations and setting up experiments, it was convenient to have a built in reset command for initial placement of the servos. Also, once reset, both camera servos possessed known positions, providing a means to quickly locate the camera in the event of a loss of power.

Once device selection had been decided, the next stage in development was to establish commands issued to any given device. As previously mentioned, devices 101, 110 and 111 have no commands associated with them, unlike the rest that all have commands based on function. The first device in Table 12 was the dc motor. The associated commands for the dc motor can be found in Table 13. The TS-5500 was able to issue any one of these four commands to drive the dc motor in a forward or reverse direction in addition to turning off or using the brake to halt the existing locomotion.

Bin.	Dec.	Shifted Dec.	Description
00000	0	0	Stop
00001	1	8	Brake
00010	2	16	Forward
00011	3	24	Reverse

Table 13: Higher 5 bits, command for dc motor

Bin.	Signed Dec.	Dec.	Shifted Dec.	Description
10000	-16	16	128	Move servo by -16 steps
10001	-15	17	136	Move servo by -15 steps
10010	-14	18	144	Move servo by -14 steps
⋮	⋮	⋮	⋮	⋮
11110	-2	30	240	Move servo by -2 steps
11111	-1	31	248	Move servo by -1 step
00000	0	0	0	Move servo by +1 step
00001	1	1	8	Move servo by +2 steps
⋮	⋮	⋮	⋮	⋮
01111	15	15	120	Move servo by +16 steps

Table 14: Higher 5 bits, command for camera servos

Next, a separate set of commands were included for the position control servos associated with the camera. Both the pan and tilt servo motors used the same set of control commands that can be found in Table 14. The table lists a number of steps moved for each command, the actual step size varied slightly for each brand of motor but on average was approximately 0.8 degrees per step. The camera servo motors were free to move anywhere within their range. To prevent exceeding the range of a servo motor, protection software was written in the firmware to avoid moving to unattainable positions potentially burning out the motors or wrecking the gears.

Bin.	Dec.	Shifted Dec.	Description
00000	0	0	Open grip
00001	1	8	Close grip
00010	2	16	Future expansion
⋮	⋮	⋮	⋮
11111	31	248	Future expansion

Table 15: Higher 5 bits, command for line grip servos

The final control commands were issued to the servo motors belonging to the line grip. To simplify matters, two commands were used for running the line grip, an open and a close command. These were intended for devices 3 and 4, implying that both parts of the line grip were controlled separately. Table 15 shows the commands used and room available for expansion to allow for various positioning of the line grip. This leaves a total of thirty additional positions available for each of the line crawling grips for future expansion. At the time of writing this report, the first two were all that was needed, allowing the grips to open and close.

This section has established the protocol that was used to communicate between the PIC and the TS-5500. All of the motors were easily managed using this standard and room was still available for expansion.

3.8 Locomotion and Position Control Motors

Description of the different types of motors used to power the line crawling robot were split into two parts, locomotion drive and position control. The motors went through several revisions throughout the evolution of the robot design based on the changing demands placed upon them. The locomotion drive will be discussed first, including the design requirements, alternative motors explored throughout the project and the final motor selection. Next, the position control motors will be discussed, the various requirements for each will be included along with a discussion of performance and alternatives where applicable.

3.8.1 Locomotion Drive

The starting point for motorizing the line crawling robot was to include a locomotion drive. The first drive that was integrated into the line grip was a standard dc motor without a gearhead. This was effective for the initial design with no payload, however the speed was too high and the torque was too low for satisfactory control and pushing any sort of moderate weight. This forced a revision into a second generation of motor drive with the introduction of a payload.

Increasing the weight due to additional control boards, power supplies and wireless communication devices required a stronger motor drive to meet the demands placed upon the line crawler. When specifying the new locomotion drive the considerations included slower rated output shaft speeds, higher torque and the introduction of a gear-head motor. The second generation, Sanyo GM-14 gear motor (gear ratio of 297.1:1) can be seen in Fig. 3.31. Unfortunately, shortly after the modifications were made to mount

Figure 3.31: Second generation, Sanyo locomotion drive installed

the Sanyo motor on the line grip, it was discovered that increased payloads were needed and the torque was insufficient to drive the line crawler once again. This was as a result of two factors, newly added weight in the payload that was not compensated for in the original torque calculations, and the efficiency lost in the gear train weakened the output strength. Both of

Gear Ratio	Gearbox Efficiency
$6:1$	81%
$30:1$	73%
$75:1$	66%
$100:1$	66%
$180:1$	59%
$300:1$	59%
$500:1$	59%
$800:1$	53%
$1000:1$	53%
$3000:1$	48%

Table 16: Gearbox efficiency vs gear ratio

these factors were corrected and compensated for in the third revision of the locomotion drive. The efficiency of the gearhead was estimated from Table 16 [9].

The latest revision of the locomotion drive was based on the discussion in Sec. 2.2.1. To provide a more robust locomotion solution, the variables that were revised included the angle of inclination of the sky wire as well as the coefficient of friction. The new estimates were strengthened to allow for the possibility of the addition of a greater payload for the line crawler in future revisions. The original estimate for θ, (the angle of inclination) was 15 degrees, this was upgraded to 45 degrees to include steeper inclines as there were no standards provided from Manitoba Hydro with regards to the angle of inclination of sky wire. The coefficient of friction, μ was originally selected at 0.4 (from a range of 0 - 1), which was based on the

early revisions of the line crawling robot. Rather than risking slippage on the skywire, the revised estimate was increased to 0.9 as the rubber-lined wheels provided extra traction, increasing the friction to get underway. The value of acceleration due to gravity, a was -9.8m/s^2, and the estimated weight of two completed robots was 3.6kg. With these values established, the equations presented in Sec. 2.2.1 were used to generate a minimum set of requirements for the locomotion drive.

The value for F_w using Eq. 2.5 gave the force of the weight causing the line crawler to slip backwards on an incline at 24.95$kg \cdot m/s^2$. The frictional force, F_f, provided in Eq. 2.4 yielded a value of 22.45$kg \cdot m/s^2$. The total applied force, F_{app} was the addition of the two, 47.4$kg \cdot m/s^2$. Next, the power (see Eq. 2.6) and angular velocity (see Eq. 2.7) were derived to provide a better picture of motor operation. The radius, r was a known quantity at 6mm for the interior of the wheels including the rubber that came into contact with the skywire. The remainder of the variables in these two equations were unknowns. One of the design decisions made for the line crawler was to keep the velocity low since many problems were avoided as a result, including smashing into other line crawlers and obstacles or moving too quickly in critical situations such as acquiring images. An angular ve-

locity of 2π radians/s was specified. This resulted in a velocity of 3.77cm/s. After substituting this value into Eq. 2.6, the resulting power requirement was 1.787 Watts. Using the value for power in Eq. 2.8, the torque necessary to drive the line crawler up 45 degree inclines was $0.28441 N \cdot m$. As most of the motor suppliers were from US origin, the value was converted to imperial units, resulting in 40.27oz-in of required torque.

Calculated values and decisions from the model provided in Sec. 2.2.1 were used to specify commercially available options. In addition to the torque and speed, the operating voltage needed to be selected. The previous motors operated from the original five volt power supply used for the controllers, however since the new torque requirements were increased, fewer low power motors were available to meet those specification. Common operating ranges for commercially available dc motors were either 6 or 12 volts. Since the line crawler was operating off regulated battery power, the lower of the two voltages was more appealing. Besides these three parameters, there were spacing constraints to consider as well. Since the line grip had already been established at this point, the most favourable option was one that did not require a complete overhaul of the grip design. The spacing available can be seen in Fig. 3.32 which represents a top view of the line

grip. The dimensions available for the new dc motor were tight considering

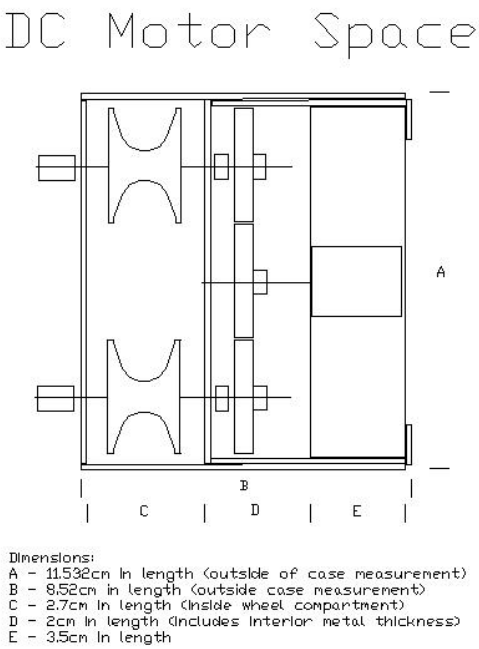


Figure 3.32: Top view of line grip space allocation for dc motor

the increased requirements. In Fig. 3.32 the important dimensions to consider were the total width of the rear compartment (approximately 4.7cm) as well as the spacing for a gear head to fit should it have hung below the upper grip (2cm in width) and finally the height that the drive shaft interfaced with the power train needed to be considered (approximately 1cm from the base of the motor was the desired value). Too much of a change in the drive shaft location would have warranted a re-design of the line grip, so it was one of the more important constraints when specifying a replacement locomotion

drive. The remaining dimensions of the motor and its weight (maximum allowable weight was 300 grams) were easier to accommodate.

Three alternatives for a locomotion drive were considered, a parallel shaft configuration, an offset shaft, and a motor with less than the required torque but with an added external gear train to reduce the speed and increase the torque. Each of these possibilities were considered in turn and based on utility, how close they matched our requirements and availability

The first configuration considered was the parallel drive shaft. This type of dc motor consisted of a gearhead with an output shaft parallel to the drive shaft, often making for a long slim construction dependent upon the type of gearing configuration. The most important attributes specified for this option included speed, power and torque. The spacing was not quite as much of a concern since parallel shaft motor shapes were well suited to the line grip. A motor available from MicroMo Electronics [37] was discovered that was close to meeting all of the specifications for performance and size. This was a two part construction, with a dc micromotor, the 'Series 2230 006S', and a gear head, the '38/3'. The micromotor weighed 50 grams and was rated with 82% efficiency. The maximum output torque was 2.5mNm and the output shaft speed was 8,000rpm. In addition to the motor ratings, the

shaft was 1.1cm from the base of the motor making it nearly identical in position to the desired height of 1cm. Adding the ’38/3’ gearhead to the motor changed the motor dimensions and capabilities. The overall length of the motor and the gearhead was 5.84cm which was greater than the maximum width in the upper compartment. However, the advantages associated with adding the gearhead made it far more appealing as the maximum output torque increased to 1200mNm, or 169oz-in. The efficiency of the gear train was rated at approximately 53% with a reduction ratio of 689:1 and the additional weight was 92 grams making for a total weight increase to 142 grams. Adding the gearhead reduced the speed to approximately 11rpm which satisfied the original maximum allowable speed constraint. The only potential problem introduced with the gearhead was the shaft height and bore size. The new shaft height with the gearhead was 1.2cm, warranting more work to lower the dc motor mounting shelf and the new bore size of 1/4” required ordering new driver gears. Taking into account the loss of performance due to the motor efficiency and the gearhead, the output torque was estimated at approximately 73.45oz-in at the output shaft which was nearly double the amount required to drive twice the weight of a single line crawler. This extra torque allowance provided some additional room to move on the motor

efficiency curve and still ensure satisfactory operation.

The second configuration for a locomotion drive was the offset shaft variety. For this type of motor, rather than having the main drive shaft operating in parallel with the motor, an offset due to the introduction of a gearhead relocates the output drive shaft away from the motor drive shaft. This posed a new mounting problem since the gearbox could be quite large and the dc motor quite small. As a result, the centre of mass and how the motor attached to the grip would likely require mounting blocks to secure it. This motor was selected based on the same requirements as the previous parallel shaft motor with the only exception related to spacing. A promising solution from Merkle-Korff Industries [34] was discovered, the size was appealing as it would fit into the compartment space available with only minor modifications. The desired model was the D47 Plastic Series DC gear motor. There were both 6 and 12V varieties available with torque ratings equal to 40in-lbs, corresponding to 640oz-in and rotational speeds of 25rpm. The efficiency and motor characteristics were not provided on the suppliers website and were estimated based on experience with other dc motors of similar size and parameters. Similar to the parallel option already discussed, a 0.25” shaft was suitable as it was a common bore size. Unfortunately, the weight

of the motor was not provided in the datasheets or on the website so it was estimated based on dimensions and materials to equal approximately the same weight as the parallel configuration.

The final alternative considered was an under-powered motor, strengthened with a gear train to reduce speed and increase torque. The extra loss of efficiency from adding a small gear ratio after the motor would be negligible. The most significant motor characteristics for this option was the torque, which could be approximately cut in half and the speed, being inversely proportional was approximately doubled. This resulted in finding motors that provided around 25oz-in of torque and speeds of 120rpm acceptable. The motor that most closely met the needs for this alternative was from Micro Drives [35], one of their dc gearmotor series, model number MD3636 + MD35C. Corresponding to the dc motor and the gear head pair. The dc motor model was the MD3636 A006V, the 6V version of this motor. The motor efficiency was rated at 70% and the gearhead selected had an efficiency rating of 59%. Together they provided an overall efficiency of 41.3%. The output torque was rated to 588mNm or 83.3245oz-in. Including the loss due to efficiency this dropped to 34.413oz-in at the output shaft, leaving some room to operate below max efficiency. An additional gear

ratio of 3:1 in the power train brought the overall output torque up to an acceptable level of 103.239oz-in. The overall weight of the motor/gearhead and extra gears was not provided but was estimated to be less than the maximum acceptable 300 gram limit. The output shaft speed was 24rpm, and the overall length of the motor was 52.5mm, excluding the output shaft. Similar to the other two alternatives, some modifications would need to be made to the upper grip to house this new motor.

Each option was considered before deciding on the parallel shaft geared-motor configuration. This selection was the simplest to install as it was closest in fitting the available dimensions. The output characteristics exceeded the required specifications in a smaller package, weighing less than half the maximum amount. The only change in gearing was a replacement driver gear with a bore size to match the new shaft dimension of 0.25”. The other motors were not without merit, but the offset shaft motor turned out to be unavailable in small quantities and since it would not have been favourable to purchase hundreds, this choice was discarded. The final option with the reduced performance requirements was also discarded due to the need of additional gearing hardware and the space requirements meant too many changes were needed to support this alternative.

When mounting the new drive, a similar form from an unfurled hose clamp was used to secure it to the mounting tray. The tray had to be lowered by 0.2cm in order for the new shaft to mate properly with the gear train. This was achieved through drilling out the holes in the chassis and using nuts and lock washers for the screws to drop the platform 0.2cm below the original value and then locking it into place with the new motor secured as part of the drive train (see Fig. 3.33). One additional point to note re-

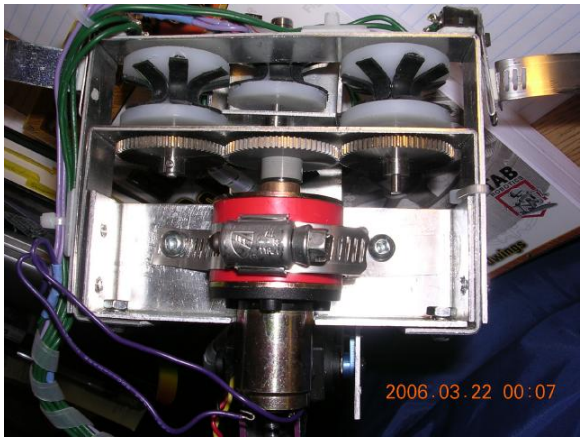


Figure 3.33: The new locomotion drive secured in place

garding the new motor install was that the problem with the overall length was solved by cutting space in the back of the chassis to allow the motor to protrude behind the line grip. This did not interfere with normal operation, completing integration of a new drive into the line grip.

3.8.2 Position Control Motors

To control position of the joints for the line grip as well as two degrees of freedom for the camera, position control servo motors were employed. Three different brands of servo motors were used with slightly different specifications for each. They included, Futaba servos for camera positioning, Hobbico servos for camera positioning and lower line grip operation and finally Hi-Tec servos for upper line grip operation. A brief discussion of position servos followed by each particular brand used and its function follows.

Servos are often small dc motors providing high torque at their output shaft due to large internal gear ratios [9] (see Fig. 3.34). They commonly have three wires, one for power, one for ground and one for a control signal. With power and ground connected and a signal present on the control wire, the intended position of the output shaft will be maintained. The refresh rate for most servo motors ranges from approximately 12-26ms. Changes in the signal on the control wire result in changes of the position of the output shaft. Length of the signal present on the control wire dictates the desired position of the output shaft. Each type of servo motor has its own range for

Figure 3.34: Hi-Tec servo motor

acceptable control signal lengths. Most servos are limited to 180 degrees of possible movement [9].

The first type of servo was used for the camera pan operation, this was a Futaba S3003 standard servo. The requirements for the panning servo were very relaxed as the weight of the camera and the tilt servo was estimated at 100 grams. Converting grams to ounces yielded 3.53oz for the camera and tilt servo. As a result, the specifications for this servo exceeded the requirements. The Futaba servo was rated to 44oz-in, more than ten times what was required and only weighed around 37 grams, it was overqualified for the desired task. The main reason for selecting this servo was that it had

the least expensive price tag in that range of performance.

The second type of servo was used for tilting the camera as well as operating the lower grip motion, this was a Hobbico CS-35 model servo. For the first task of tilting the camera, this had even less stringent requirements than the Futaba servo used for panning the camera as it was only required to support its own weight and that of the camera (less than 100 grams). The range of motion required for the tilting servo was 90 degrees due to the shape of the camera. The output torque at the shaft was rated for 54oz-in, and the overall weight of the servo was 27 grams, making it an even lighter alternative than the Futaba servo. The reason that this servo was not used for panning as well was due to the cost as it was almost double the price of the Futaba standard servo. The second task of operating the lower line grip required slightly more torque however still nowhere near the output torque provided by the Hobbico motor. With a weight of approximately 150 grams, the Hobbico servo provided over three times the necessary torque to drive the lower grip up and down over a 90 degree range.

The third and final type of servo was used for operating the upper line grip motion, this was accomplished with the Hi-Tec HSR-5995TG. Based on the design of the line crawling robot, the upper grip servo motor was

responsible for supporting the bulk of the weight. As a result, the servo selected for the task had much higher ratings. The estimated weight for a single line crawler was approximately 1.6kg. The maximum output torque was rated at 417oz-in. Working backwards, this implied that it could support 3.6kg, slightly more than double the weight of a line crawler. In addition to the much improved output torque, the gears inside were all constructed from titanium alloys, providing a longer lifetime compared to the plastic gears found in the other servo motors. The overall weight of the Hi-Tec servo was 62 grams, adding very little to the line crawler and providing a strong workhorse for upper grip support during operation. This servo was selected based on its performance. At the time the line crawler was being built it had the highest output torque rating available, making it the most suitable selection (see Fig. 3.34).

This concludes the discussion of motors selected to control position of the line grip, the camera and the locomotion drive. After selecting and integrating these five motors, various positions were obtainable for both the grip and the camera and the locomotion drive was strong enough to scale inclines of 45 degrees providing motion in either a forward or reverse direction at acceptable speeds for safe operation. Further details on motor control

are provided in Sec. 4.4.

3.9 Power Supply Design

Another key part of the design of the line crawling robot was the power supply. A couple of different regulator designs were explored as well as the idea of multiple supplies to improve the range of the robot. There were a number of considerations including estimated load demands, various device voltages, minimizing space, and weight limitations.

Two types of power regulators were investigated to compare suitability for the line crawler, linear and switching regulators. Linear regulators have an advantage in the form of being simple in construction and operation, but with the drawback of poor efficiency [16]. Often they consist of a single regulator component. Additional stability capacitors and a heat sink may be necessary depending upon the situation. For setups with large current draws, heat sinks will almost always be required. This adds more weight, takes up more space and in turn requires more power to haul around the weight. The efficiency of linear regulators was the main disadvantage as it is rarely very high. For the case of supplying power to the TS-5500 at a nominal voltage of approximately 1 amp, the efficiency would be at best

around 30 to 40% [16]. Since the design was meant for a mobile robot with a limited power source, this was a concern. The initial power supply during experimental stages made use of a linear five volt regulator (a 7805 device) including a heatsink. The average lifetime of the supply when using 8 'AA' batteries (rated at 2200mAh each) was approximately two hours. This was sufficient for running shorter experiments, but posed a problem for extended operation.

The alternative approach investigated used a switching regulator. The main advantage of switching regulators over linear ones is a much greater efficiency in power transfer, data sheets often claim efficiencies of 80% or higher [40]. The reason for such improved performance is related to the output duty-cycle, for a more in depth discussion, see Sec. 2.2.2. This large improvement in efficiency is appealing for an autonomous design with limited power available. Switching regulators do come with disadvantages as a price for the improved efficiency. The circuitry required is commonly more complicated and heavier than that of linear regulators [16]. Also, switching regulators are inherently noisy and can cause electro-magnetic interference (EMI) problems [16]. To help limit the EMI problem, switching circuit designs need to be kept as compact as possible [16, 40]. There are several

types of common switching regulators, the Buck, Flyback (or Buck-Boost), and variable switching regulators. The line crawling robot needed a regulated five volt supply and the onboard battery packs provided 9.6 volts dc. Since the regulator needed to step down from a higher input voltage to the regulated supply voltage, the Buck configuration was most suited for the task [16].

The schematic of a buck regulator is shown in Fig. 2.4. The switching controller is provided in a complete package from manufacturers resulting in a need for only a few discrete components to complete the design. Two capacitors, a diode and an inductor were necessary. The component values and the reason for selection follows. The input capacitor, C1 was $100\mu F$ electrolytic capacitor rated for 35 volts, selected with over three times the maximum input voltage rating and at a value that provided input stability to the regulator [40]. Diode, D1 was a 3A Schottky diode (1N5820), selected for quick response to provide a return path for current stored in the inductor and rated as high as the maximum output current [40]. The output capacitor, C2 was $1000\mu F$, 25 volt electrolytic capacitor, selected to filter the output smoothly and the voltage rating provided an improved equivalent series resistance (ESR) to help reduce the output ripple [40] The inductor

L1 was selected as $68\mu H$ rated for 3A, the value was kept low to operate in constant current mode with a current rating able to support the maximum output from the regulator [40]. The switching controller selected was a National Semiconductor LM2576, TO-220-5 package with a 3A rating. The frequency of the switching regulator was internally fixed at 52kHz. This design provided enough current for the heavier processing requirements of the TS-5500 during learning, image processing and more complicated sensing routines. In order to reduce the chance of any ground loops, EMI, or wiring inductance, the leads of the devices and connections were cut as short as possible. When laying out the devices for construction, this resulted in a compact perf-board size of 4cm x 3.5cm. The finished product can be seen in Fig. 3.35. From examining Fig. 3.35, there are only four visible compo-

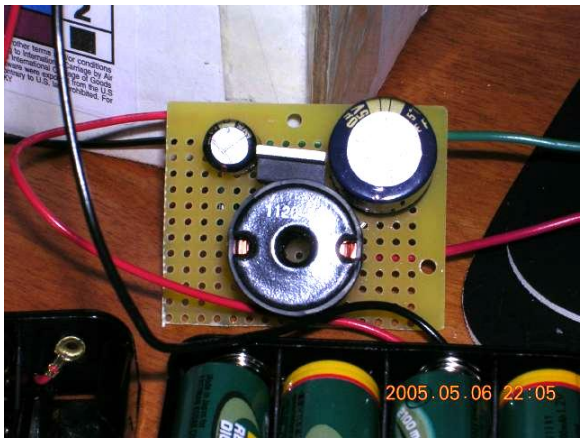


Figure 3.35: Completed buck regulator

nents because the catch diode was installed underneath the board.

With the regulator design complete, the next step was integration into the system, replacing the preliminary linear regulator. The linear regulator provided approximately 2 hours of operation for the line crawler. Changing to the switching regulator doubled the life of the batteries, yielding over 4 hours of continuous operation. When the newest locomotion drive from MicroMo was installed into the line crawler, the power needs of the robot were addressed again. The TS-5500 was known to draw approximately 1 amp of current during regular operation and the PIC control board current draw was rated at 3mA, almost negligible in comparison. The motors and sensors were rated to collectively draw about 800mA of current, to account for any fluctuations or operation under extreme conditions, an estimated value of 1 amp was chosen. The devices were separated as follows, the TS-5500 was powered from one regulator and the PIC, the RS232 transceiver, the h-bridge chip, the motors and sensors were powered from a second regulator. The pros of having separate power sources and using the extra weight to help balance the payload outweighed the cons associated with additional weight. Rather than creating a new design and since the power requirements were almost identical, the existing buck regulator was duplicated and an ex-

tra battery pack was added to the robot (see Fig. 3.36). This helped boost the

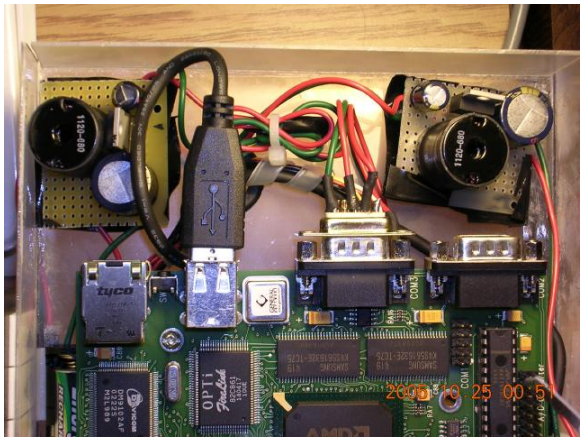


Figure 3.36: Two buck regulators onboard the line crawler

life of the line crawler to a minimum of 4 hours regardless of the amount of onboard processing. There was one exception to the use of regulated voltage inputs and that was for the locomotion drive (as discussed already in Sec. 3.8.1). Both the intelligent h-bridge IC and the dc motor were driven from the raw dc voltage of the battery pack (9.6 volts). All other remaining devices were powered by the five volt buck regulator.

3.9.1 Battery Selection

The next step was to select batteries to power the line crawling robot. This section includes a brief look at the power requirements, limitations, the products available and the final decision for power solution.

As mentioned previously in Sec. 3.9, eight 'AA' batteries were used to power the line crawler. The voltage regulators provided a five volt output from a step-down voltage of 9.6 volts (8'AA' batteries) and the power requirements were estimated at approximately 1 amp during heavy operation. The restrictions applied when specifying a set of batteries consisted of space constraints, weight allowance, price, availability, product lifetime and charge density. With the intent of providing a degree of autonomy to the line crawler and to reduce cost and waste, rechargeable batteries were the only variety considered.

The four types of batteries examined included lead-acid, nickel-cadmium (Ni-Cad), nickel-metal-hydride (NiMH) and lithium-ion (LI). The lead-acid batteries proved to be too bulky for the space and weight constraints. The remaining three options were available in suitable sizes, and weight. At the time of specification, the price and availability of the LI cells were more expensive and harder to acquire, leaving only two types of batteries.

The nickel-based cells maintained popularity for consumer electronics, making them widely available and relatively inexpensive. The Ni-Cad batteries were favourable as they provided the cheapest alternative with an estimate life cycle of up to 1000 charges [8]. However, the maximum charge

density available during the specification was 1000mAh, implying that the batteries would have supplied enough power for 1 hour of operation. Also, the Ni-Cad cells were susceptible to a $memory - effect$ more predominantly than the NiMH batteries [8]. This referred to the capacity of the battery being reduced over time due to development of crystals on the cell plates [8]. In order to reduce the likelihood of developing this problem, regular, complete discharge of the Ni-Cad cells was needed [8]. A complete discharge of battery power would have caused problems for the line crawler since the position control motors supporting the line grip would let go in unpowered conditions, potentially causing disaster. The NiMH cells offered much greater charge density, rated up to 2500mAh, but with a reduced life cycle, estimated around 300 charges, and a more rapid rate of discharge when unused [8]. The availability and cost of both nickel-based batteries were very similar. As a result, taking into account the trade-offs and the fact that the Ni-Cad batteries contained much higher concentrations of toxic metals, the type of cells chosen were NiMH. The extra charge-density, lower susceptibility to the $memory-effect$ and the more environmentally-friendly construction were the highlights considering the remaining system requirements were met by both varieties.

3.10 Sensor Configuration

The sensor configuration of the line crawling robot provided a means for interaction with its environment in both a reactive and proactive manner. There were a few different sensors employed by the robot, including contact switches, infrared (IR) sensors and a camera system (see Sec. 3.4 for more information regarding the camera). A discussion of the sensors, their placement, and use are included in this section.

The first set of sensors added to the robot were contact switches. As their name implies, they were intended to detect contact with external objects during locomotion. Four sensors were placed around the line grip to allow for detection of any obstacle near the work envelope. A 3 amp lever micro-switch was used for all of the contact sensors (see Fig. 3.37). The lever on the switches covered little surface area and was not easily triggered by obstacles so custom extensions were fabricated and fitted to each. The extensions were created in pairs, two for the upper grip and two for the lower grip. Each of the extensions was formed to help broaden the contact area for triggering the switch upon contact with the types of obstacles encountered on the skywire. The upper grip switches were meant to detect

Figure 3.37: The microswitch used for contact sensors

line clamps or obstacles that extended above the wire whilst the lower grip switches were intended to trigger when vibration dampers or obstructions hanging below the skywire were encountered. A conceptual drawing of the first revision of the upper and lower extensions for the contact switches can be seen in Fig. 3.38. The sensors were placed in pairs on either side of the upper and lower grip to cover both directions of travel. The micro-switch extensions were cut and formed from a standard three inch hose clamp, similar to the material used to secure the locomotion drive. The shape for the upper grip extensions were formed pointing out and downward from the upper line grip. The intention was to have contact with the line clamp or any

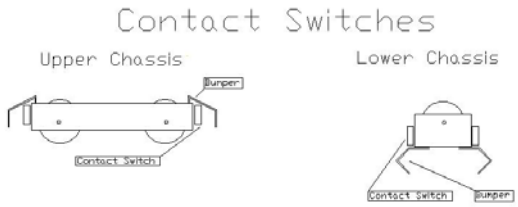


Figure 3.38: First revision of extensions for contact switches

obstacle extending above the skywire hit the extension first, triggering the micro-switch. The switch was connected to the upper line grip with high strength foam tape and centered over the direction of travel to ensure the best possible angle of contact with any obstacles in its path. The lower grip contact switches were placed upside down on the lower chassis so that the extensions would push up and trigger the micro-switches during encounters with the vibration damper or obstacles encountered beneath the sky wire. The lower extensions were similarly formed outward and away from the contact switch to fall into the path of obstacles hanging below the skywire. After some preliminary testing both the upper and lower switch extensions were revised to provide a more sensitive contact during obstacle encounters. The revised components are shown in Fig. 3.39. The upper extensions were lengthened and added increased flexibility to allow for a more sensitive touch as opposed to the more rigid original design. The lower exten-

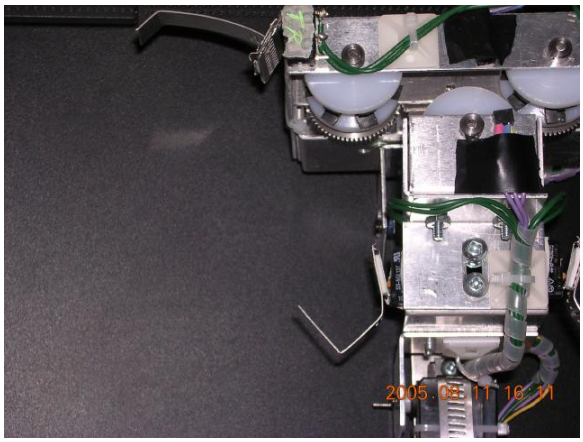


Figure 3.39: Second revision of extensions for contact switches

sions were inverted and lowered to allow contact from beneath to trigger the switches more readily as it was discovered during verification that facing the other way was difficult to trigger the switch. Both the upper and lower extensions remained centred over the skywire for their respective direction of travel to provide immediate notice if and when obstacles were encountered.

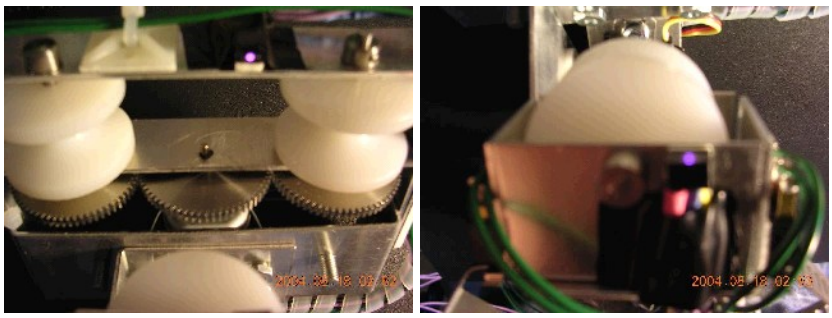
Next, infrared sensors were added to the sensory plan as part of the line grip control layer. The main function was to provide a means of feedback regarding position of the upper and lower line grip assemblies with respect to one another. Although the position servos adjusted the shape of the line grip to allow it passage around obstacles on the skywire, the possibility of attempting to close over an obstacle or blockage would not have been registered except internally to the servo electronics. A quick verification of

line grip position using the infrared sensors provided a means of discovering whether the grip had closed properly or not.

The Fairchild Semiconductor, QRD1114 IR Reflective sensors came in pairs, both the upper and lower grip were outfitted with an IR sensor. The location was chosen at the front of the upper and lower chassis modules on the outside, with the transmitter receiver pair facing down on the upper chassis and up on the lower chassis. The mounting consisted of 2-sided foam tape to mount the sensors to the line crawler with additional mounting tape wrapped around the outside to protect and secure them from damage due to external forces.

The IR sensors were intended to be used as a feedback tool during operations that required the line grip to open and close. When the grip opened, the IR beam from the two transmitters would cease to point at each other, registering increased sensor values, indicating that the grip was open. Conversely, when the line grip closes, the IR beams pointed at each other again, returning the sensor value to its original intensity, indicating a successful grip closure. In the event that an obstacle prevented the line grip from closing properly, the IR sensors would indicate a problem when polled and provide feedback that re-alignment was needed to solve the problem. The lo-

cation of the two IR sensors can be seen in Fig. 3.40, the light emitted from the transmitters had infrared wave-lengths which are normally invisible to the human eye, but the CCD (charge-coupled-device) of a digital camera was able to sense IR and displayed it as a light purple or violet colour. For more details about the IR sensors, see Sec. 4.5.1



(a) Upper grip IR sensor (b) Lower grip IR sensor

Figure 3.40: Infrared sensors, position and alignment

The final sensory input used by the line crawling robot was the camera. A detailed discussion of the camera selection and implementation can be found in Sec. 3.4. From a sensory perspective, the camera was used to aim at and acquire images of specified targets. In addition to these devices, proximity sensors were investigated as a means to reduce potential vibration damage to the robot by sensing approaching obstacles and taking action before the contact switches triggered. Unfortunately, at the time of writing

this report, the proximity sensors had not yet been implemented. This completes the discussion of the sensory equipment configuration on the ALiCE II robot.

3.11 Reinforcement Learning and the Target Tracking Problem

The reinforcement learning process discussed in Sec. 2.3 contained three main parts, states, actions and rewards. This section of the report discusses the target tracking problem, and shows how the three parts were developed around the problem, algorithm selection including a section discussing the rough coverage modified versions of each and any additional changes necessary for their operation. Followed by a brief discussion of the classical target tracking algorithm that was used as a baseline for comparison with the learning methods.

3.11.1 The Target Tracking Problem

The starting point for developing a RL system was the problem definition. The goal of the target tracking system was to lock onto a target and track it during movement, keeping it centered in the field of view for the camera at all possible times until the target was no longer needed. This translated into

a useful behaviour for the ALiCE II platform, being able to track objects of interest to gather meaningful images from several vantage points during travel. The stages in developing the reinforcement learning environment included defining the possible states, outlining actions to be taken in any given state and generating reward values.

States were based on two components, image size and how it was separated into sections. To keep the processing requirements as low as possible, the initial image size was selected at 160x120 pixels, then converted to greyscale, and further decimated by a factor of 4, yielding a 40x30 greyscale pixel image. During tracking, a target could be located anywhere in the field of view consisting of the 40x30 image, subsections were created to provide further knowledge where the target was in relation to the centre. Images were divided up into 9 parts, as seen in Fig. 3.41 with each state labeled S0 through S9 consisting of a block of pixels. The goal state, S4 was the target as it centred the camera's field of view, providing the best possible images. Each state had a corresponding set of actions for re-positioning the camera as needed to support the goal of centring the camera view on the target.

Actions were selected based on the location of the target with respect to the field of view. The camera was able to distinguish between nine different

Figure 3.41: System states

regions (states) shown in Fig. 3.41. In order to move toward the goal state, actions were defined that provided a number of steps to move the servos in the direction of the target. The states belonging to the 4-neighbour configuration only required a single direction of travel while the remaining 4 states from the 8-neighbourhood required both vertical and horizontal movement. The directions of movement for any given state were setup with h representing the horizontal or panning motion and v representing the vertical or tilting motion (see Fig. 3.42). These divisions in Fig. 3.42 coincided with the states from Fig. 3.41. For example, this implied that for state 0, the motors activated will be both the horizontal and vertical servos. Positive numbers indicate that the direction of travel was made with positive step increments

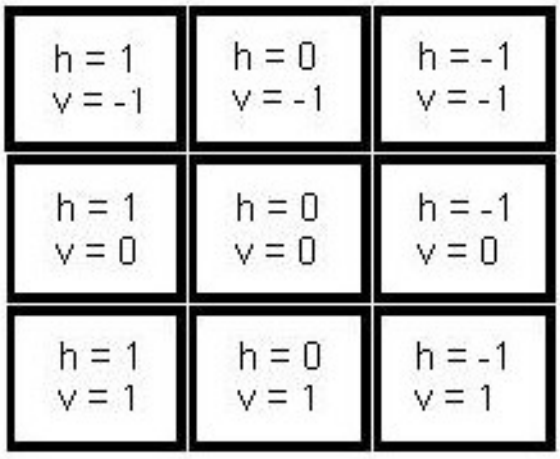


Figure 3.42: Directions taken pertaining to current state

toward the goal and negative numbers indicate the the action was provided with negative step increments to achieve the goal. There were a total of twelve possible actions for any given state. Limiting the amount of available actions was done to restrict the size of the RL problem and demands placed on the TS-5500 during the learning process. The twelve actions consisted of step increments ranging from 0 to +/-11. Each step increment or decrement to the servos provided a rotation of approximately 0.8 degrees in the selected direction with an accuracy of +/-0.25%. Once the actions had been established, it was important to devise a means of rating performance of the actions, which leads into a discussion of the rewards.

The reward function varied from 0 to 1, with 0 representing the worst

case and 1 indicating the goal state. To create a range of rewards from any position in the camera view, a Euclidian distance metric was used to measure from the target (x,y) to the centre of the field of view (0,0). Since the centre was at the origin, the distance calculation simplified to Eq. 3.1.

$$distance = \sqrt{x^2 + y^2} \tag{3.1}$$

The reward function represented in Eq. 3.2 shows how the current distance divided by the maximum distance subtracted from 1 provided a unique normalized measure corresponding to anywhere in the camera field of view.

$$reward = 1 - \frac{distance}{maxdistance} \tag{3.2}$$

The maximum distance is the worst case, which will yield a reward of zero and for anything less than the maximum distance the reward becomes progressively better until it reaches one. This occured when the current location coincided with the origin of the image (0,0). With the RL framework in place, the next step was to select the algorithms.

3.11.2 Algorithm Selection

The next important stage in developing the RL solution for target tracking was selecting algorithms suited for the task. Constraints relating to the portable nature of the ALiCE II platform were considered. First, processing power needed to be minimized as the more computations required, the less lifetime the batteries would provide. Also, for a real-time problem where actions needed to be resolved and processed quickly, faster algorithms would provide more favourable results. A number of different methods were considered when selecting the most suitable algorithms.

The temporal difference (TD) algorithms were most suited to the requirements as they met both criteria. Using single-step TD methods that took advantage of bootstrapping, less processing power was required and learning with bootstrapping implied that each step revised the algorithms perception of its environment during regular operation [67]. Alternate methods were examined, including Monte Carlo (MC) methods, and eligibility traces which allowed TD methods to look further into the future similar to MC methods. MC methods did not fit as well for the RL problem proposed here as they were non-bootstrapping methods, implying that they did not

revise their view of the environment until after an episode, making it less appealing for real-time learning in a dynamic environment. Also, gradient descent methods and a planning algorithm were considered but both of these cases exhibited heavier computational requirements, disqualifying them as well. The end result favoured the TD methods, and several algorithms were selected for comparison, including the actor-critic method, Q-learning and sarsa.

3.11.3 Rough Coverage Modification

A variant of the RL algorithms was also included in the system architecture. The algorithms were modified to add approximation-space based refinement of the RL process. Specifically, the step size parameter which adjusted the learning rate of the algorithms was adjusted during the learning process. This was achieved through gathering data from each step in an episode and storing it in an ethogram-like table [71]. At a given point in any episode (usually at the end), the stored data was divided up into blocks based on behaviours. Comparisons between current behaviour and previously known acceptable behaviour provided the degree of rough coverage present in each block. Averaging the rough coverage values from all blocks

provided a feedback metric for performance, indicating how well the current set of behaviours were performing compared to previously known acceptable behaviour. The value of the rough coverage, or ν was then subtracted from 1 and multiplied by the learning rate, α, providing an adaptive learning rate that adjusted to any given performance. In the event that the system was performing well and the rough coverage value was high, then the adjusted value of α would become small with the intent of avoiding overshooting the desired behaviour. Should performance have changed for the worse, the rough coverage metric value would be smaller, resulting in a larger value for α, increasing the step size in the hopes of moving toward more acceptable behaviour.

This modification was made to all three algorithms and then compared to the classical versions. The revised formal algorithms including rough feedback are shown in Alg. 4, Alg. 5, and Alg. 6 demonstrating the changes made.

The differences of the modified versions of the RL algorithms began with initializing the value for the average rough coverage, ν, starting at zero. Also, the data table (ethogram) was cleared at the beginning of each episode, to allow for gathering current behaviour. The episode lengths were variable,

Algorithm 4: The Q-Learning Method With Rough Feedback

Input : States, $s \in S$, Actions $a \in A(s)$, Initialize Q(s,a), $\bar{\nu}$, α, γ, π to an arbitrary policy (non-greedy);
Output: Optimal action value Q(s,a) for each state-action pair;
while *True* **do**
  for $(i = 0; i \leq \#ofepisodes; i++)$ **do**
    Initialize *s* and data table;
    Choose *a* from *s*, using policy derived from Q;
    for *Repeat(for each step of episode):* **do**
      Take action a; observe reward, r, and next state, s';
      Record state, action, and associated reward in data table;
      Q(s,a) $\longleftarrow$ Q(s,a) + $(1 - \overline{\nu})\alpha[r + \gamma max_a Q(s', a') - Q(s, a)]$;
      s $\longleftarrow$ s'; a $\longleftarrow$ a';
      until *s* is terminal;
    end
    Generate $\bar{\nu}$ from results recorded in data table for current episode;
    Update new value of $\bar{\nu}$;
    Clear data table;
  end
end

Algorithm 5: The Actor-Critic Method With Rough Feedback

Input : States $s \in S$, Actions $a \in A(s)$, Initialize α, γ, $\bar{\nu}$.
Output: Policy $\pi(s, a)$ responsible for selecting action a in state s.
for $(all\ s \in S, a \in A(s))$ **do**
    $p(s, a) \longleftarrow 0$;
    $\pi(s, a) \longleftarrow \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}}$;
end
while *True* **do**
    Initialize s, data table;
    for *Repeat (for each step of episode)* **do**
        Choose a from s using $\pi(s, a)$;
        Take action a, observe r, s';
        Record state, action and associated reward in data table;
        $\delta = r + \gamma V(s') - V(s)$;
        $V(s) \longleftarrow V(s) + (1 - \bar{\nu})\alpha(r + \gamma V(s') - V(s))$;
        $p(s, a) \longleftarrow p(s, a) + \beta\delta$;
        $\pi(s, a) \longleftarrow \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}}$;
        $s \longleftarrow s'$;
    end
    Generate $\bar{\nu}$ from results recorded in data table for current episode;
    Update new value of $\bar{\nu}$;
    Clear data table;
end

Algorithm 6: The Sarsa Method With Rough Feedback

Input : States, $s \in S$, Actions $a \in A(s)$, Initialize Q(s,a), α, γ, $\bar{\nu}$, π to an arbitrary policy (non-greedy)

Output: Optimal action value Q(s,a) for each state-action pair

while *True* **do**

  for $(i = 0; i \leq \#ofepisodes; i++)$ **do**

    Initialize *s* and data table

    Choose *a* from *s*, using policy derived from Q

    Repeat(for each step of episode):

    Take action a; observe reward, r, and next state, s'

    Record state, action and associated reward in data table

    Choose action a' from state s' using policy derived from Q

    Q(s,a) $\longleftarrow$ Q(s,a) + $(1 - \bar{\nu})\alpha[r + \gamma Q(s', a') - Q(s, a)]$

    s $\longleftarrow$ s'; a $\longleftarrow$ a';

    until *s* is terminal

  end

  Generate $\bar{\nu}$ from results recorded in data table for current episode

  Update new value of $\bar{\nu}$

  Clear data table

end

but averaged around 5000 steps each. In order to reduce the amount of computation required for generating $\overline{\nu}$, the amount of steps included in the data table was restricted to approximately 20% or 1000 steps out of each episode. For the purpose of the work discussed in this report, the first 1000 steps in each episode were selected for processing.

Besides storing information in an ethogram at each step of an episode for the first 1000 steps, additional work was required to generate the average rough coverage value, $\overline{\nu}$. At the end of every episode, all of the processing power required for generating the rough coverage values and then averaging the results took place. As a result, slower performance was anticipated in a somewhat cyclical fashion due to the nature of the timing for the computational requirements when compared to the standard RL algorithms.

3.11.4 Classical Target Tracking

The classical target tracking algorithm was employed to provide a basis for comparison with the RL methods. Unlike the other algorithms, no learning took place during the tracking process with this algorithm. As a result, there were no rewards, and the states and actions differed. Instead of having states relating to subsections of the image, each pixel was treated as a separate

state, implying that when the target was located at any given pixel, tracking the centre of the field of view to that pixel was the goal. Rather than having a range of actions available for movement, the tracking procedure moved to the exact location of the target within the field of view. This corresponded to a deterministic or greedy approach. The formal algorithm is shown as Alg. 7

Algorithm 7: Classical Target Tracking

Input : States, $s \in S$, one state for each possible set of target coordinates;
Output: Deterministic Policy, π(s);
while *True* **do**
    get current state; (coordinates of target)
    pan $\longleftarrow$ target's horizontal distance from centre of camera view;
    tilt $\longleftarrow$ target's vertical distance from centre of camera view;
    move servos by (pan,tilt);
end

The author would like to extend his thanks to M. Borkowski for writing the classical target tracking method code and its associated verification of operation.

3.12 Robot Behaviour

The robot behaviour followed the main idea behind Brooks' subsumption model [5], having multiple goals within a hierarchy. The primary goal of the line crawling robot was to acquire images of power line equipment, with

secondary goals relating to avoiding obstacles and survival. Although the main purpose of the line crawler was to gather images of hydro equipment with the intent of spotting potential faults, the secondary goals were equally sensitive and on occasion more important. Survival of the robot became most important in threatening conditions such as high wind or encountering obstacles and would sub-sume control of the robot to circumvent the threat before allowing any more pictures to be taken. Diagrams of the two layers are quite similar to those shown in Sec. 2.6. The differences occur in the arrangement of the hierarchy as well as some of the sensory devices. The layer 0 control behaviour for robot survival can be seen in Fig. 3.43. This

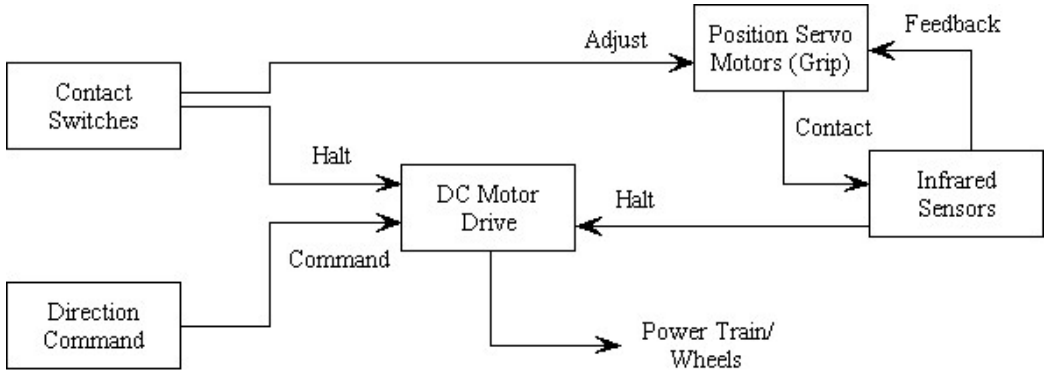


Figure 3.43: Layer 0 control behaviour for robot survival

layer consists of low level control for the locomotion drive, the position servos and all of the low level sensors including the contact switches and the infrared sensors on the line grip. A second layer, (layer 1) in the hierar-

chy introduced the camera and how it was able to sub-sume control of the robot in situations when it found a target rich environment (see Fig. 3.44). During occasions when targets were available for capturing images, the con-

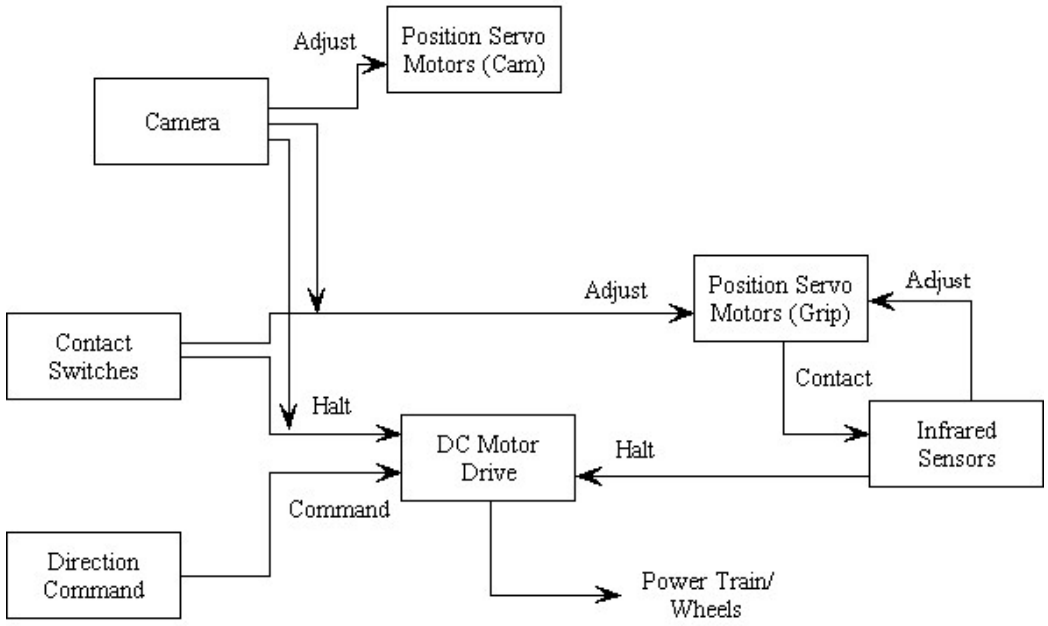


Figure 3.44: Layer 1 control behaviour for acquiring images

trol layer allowed the camera to indicate that the motors should stop any current action to allow images to be gathered. For almost all cases shutting the locomotion drive off would not cause a threatening situation. Although layer one is hierarchically above layer zero, all of the same information is available to the controller. As a result, in the event of a hazardous situation, all sensory information would be available to allow for the proper behaviour

to occur.

This concludes the outline of the basic behaviour for the line crawling robot. The subsumption architecture was useful during development of the system since as sensors and equipment was added, extra layers of control could also be added to operate in conjunction with the new devices. However, control of the more detailed behaviour associatd with target tracking and acquiring images was added separately through the use of a learning agent as discussed in Sec's. 5.3.1, 5.3.2, 5.3.3, 3.11.3, and 2.3.

4 System Verification

This section covers a description of the implementation details of the line crawling robot software and hardware-related systems. First, class diagrams outlining the software system will be presented, followed by implementation discussions for each relative section. Then a discussion of the testing interface followed by verification and optimization tasks will round out this section of the report.

4.1 System Overview

For the software systems onboard the TS-5500, the chosen implementation platform was C++ and the development environment was gcc, version 3.2.2 20030222, provided with Linux [29]. C++ was selected as the development language due to its speed, portability and familiarity. In addition, code written for the PIC was implemented using CC5X, a free C-compiler tailored specifically for PIC processors with reduced instructions and limited memory [24]. The resulting object code from CC5X was compatible with the Microchip MPLAB integrated development environment (IDE) which was used to generate the machine code and program the PIC devices [38]. Se-

lecting C for the development language of the PIC was an easy choice as it was quick to pick up, offered fast speeds, easy translation from a high level language (C) to machine code through the use of Microchip's own custom development software, MPLAB. In addition to these software development packages, Matlab was used for data analysis and output plots as it offered a wide range of analysis tools and quick times from input to generating results. The main goals of designing the software was to keep the code as compact and efficient as possible whilst providing a flexible environment that was easy to use for experimental work.

For convenience, the class diagram was broken up into three separate images to provide a picture of the software system for the line crawling robot. Although there is some overlap, the images were each focused on a specific theme including target tracking, reinforcement learning and the PIC interface.

First, the class diagram for the target tracking procedure is shown (see Fig. 4.1). Although the framework is quite extensive, my involvement for writing code was limited to the methods discussed in the upcoming sections. The remaining pre-existing framework was written by members of the CILab [3] and for the case of the camera driver, adapted from a public

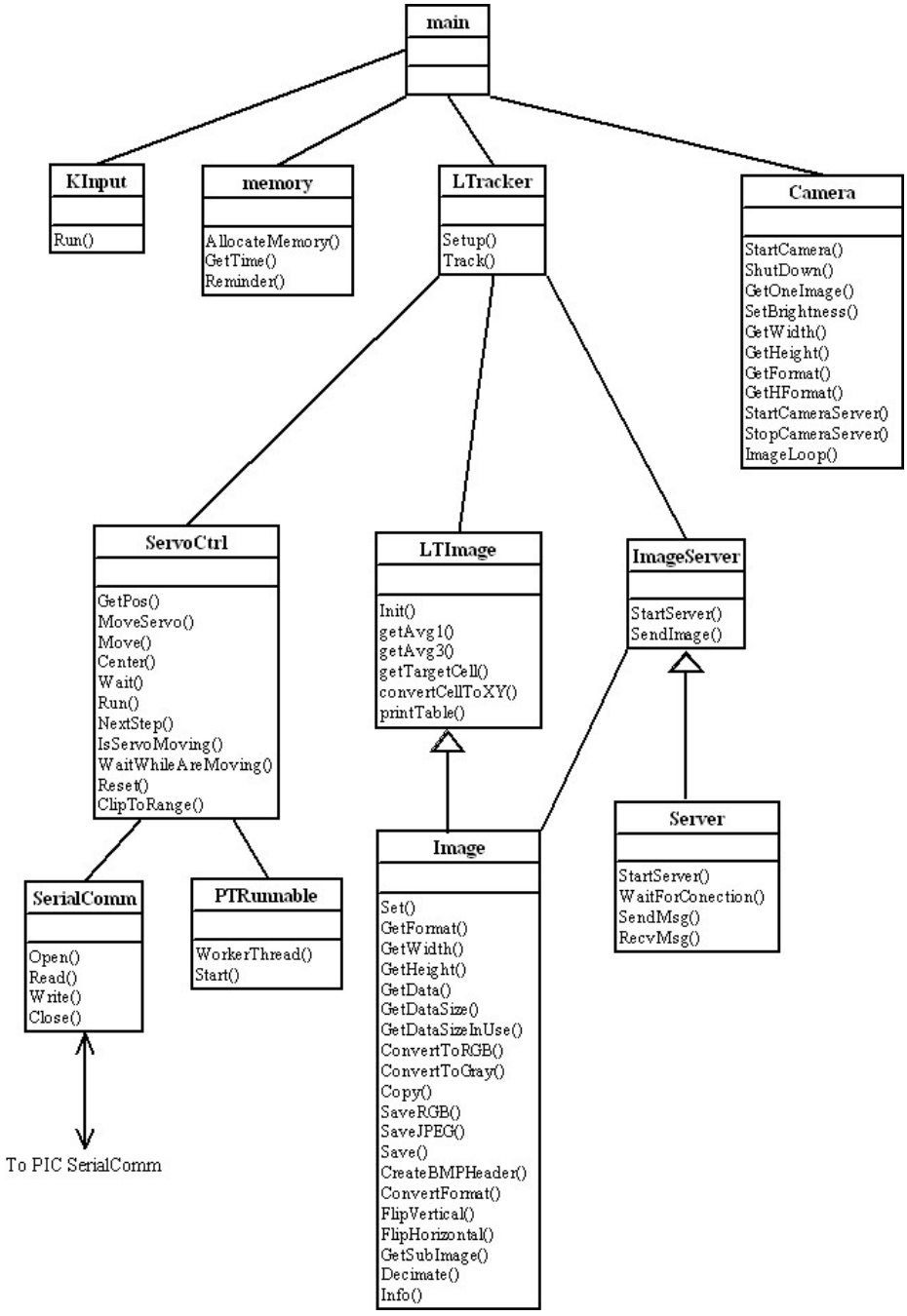


Figure 4.1: Class diagram for the target tracking task

domain source [79]. The same applies for the next class diagram, containing the structure for the reinforcement learning methods (see Fig. 4.2). Due to space requirements, only one RL algorithm was shown in the class diagram, however, the framework was such that alternate algorithms were swapped in as needed for experimental work. Also, these two diagrams show different trackers as well, with Fig. 4.1 employing the average grey level tracking method, or LTrack, and Fig. 4.2 demonstrates the alternative CTrack which corresponds to the template matching method. The code written for the first two class diagrams was contained onboard the TS-5500 whilst the final diagram represents code for the PIC controller (see Fig. 4.3).

These three class diagrams cover the software framework employed to operate the line crawling robot, including both the TS-5500 and the PIC controllers. A more in depth discussion of the software as it relates to the robot and the hardware systems follows in each of the related sections.

4.2 Target Tracking System

The purpose of the target tracking system was to lock onto a specific type of target and then to maintain it in the field of view as close to the centre as possible to capture the best possible images. There were two techniques

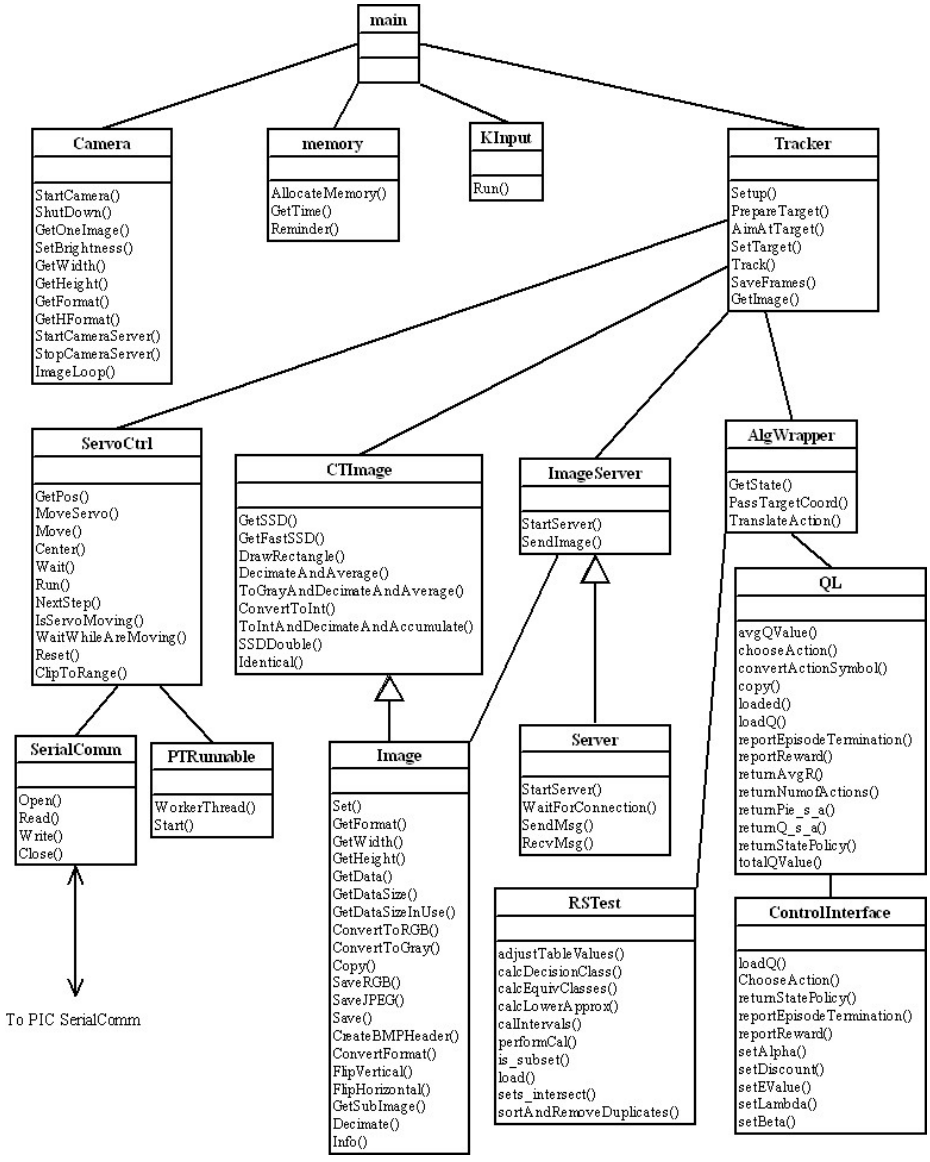


Figure 4.2: Class diagram including reinforcement learning methods

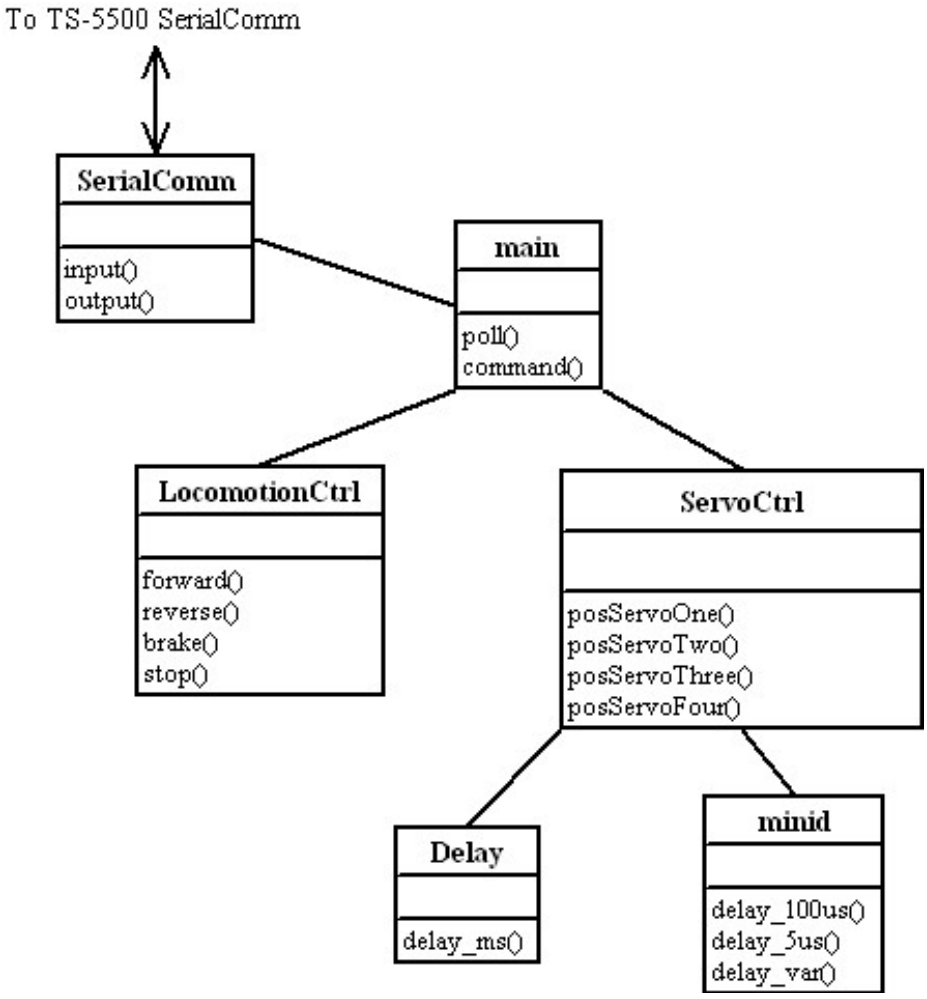


Figure 4.3: Class diagram for the PIC controller

implemented for tracking as can be seen in the class diagrams, the LTracker method as seen in Fig. 4.1 and the Tracker method as seen in Fig. 4.2. The LTracker and LTImage classes were part of the work developed for this thesis.

The purpose of the LTracker class was to centre the field of view of the camera on a desired target based on average grey levels found in its field of view. For experimental purposes, the lowest average grey level was considered the target or the centre of the target (more detail can be found in Sec. 3.4.1 and Sec. 2.5.2). This was achieved through receiving an image from the camera and separating it into subsections dependent upon the overall dimensions of the image. Pre-processing of the images included converting to greyscale and decimating by some factor to reduce the necessary work required to track the target. Once an image was received, the LTImage class provided support functions for processing and locating the target. In the event that the target was not in the goal state S4, appropriate movement commands were issued to the servos to relocate the camera toward the goal state. Before repeating this process and acquiring a new image, the tracker always waited until the servos completed current movements to ensure that the previous tracking procedure was completed to avoid compromising the

process.

The purpose of the LTImage class was to extend the existing Image class and provide support for the LTracker method. Specifically, the dimensions of an input image were discovered and then a corresponding set of subsections were chosen based on the size. The larger the image, the more subsections were available. In addition to deciding on the number of subsections, the average grey levels were generated and the subsection containing the target was discovered. The value of that subsection was then converted into an (x,y) coordinate before relating it to a corresponding state from the RL problem (see Fig. 3.41). A number of diagnostic methods were also included in the LTImage class as well to provide performance feedback and visual confirmation of the target location in a text-based format requiring minimal processing and bandwidth during system verification to ensure proper operation.

Verification of the average grey level target tracking process involved using a sample dark insulator on a light background (see Fig. 2.11). The preliminary version of the average grey level tracking method was set up to move in single steps toward the target cell without using any of the RL framework. As a result, moving the camera away from the target resulted in

tracking the opposite direction back toward the target. All eight directions were tested to ensure that tracking was possible for both the pan and tilt servos individually and together in both negative and positive directions.

The average grey level tracking method was able to track a dark target on a light background, making it reasonable to assume that integrating it into the existing framework with the RL methods would result in similar performance. This was acceptable since interaction between the tracking method and the RL framework consisted of locating the target and passing its coordinates. Instead of the tracking method moving the servos, that responsibility passed to the RL methods.

The remaining classes in the tracking diagram consisted of the framework already in place [3]. As a result, the next step in examining code developed for the line crawling robot in conjunction with target tracking includes a discussion of the learning methods used to control camera positioning.

4.3 Learning Methods

The second diagram, Fig. 4.2 shows how the learning algorithms fit into the big picture. The template matching tracker is shown in this diagram to

demonstrate how the modules were interchangeable. The same applied for the learning algorithms. Although the algorithm shown is Q-learning, other algorithms were easily substituted in its place during the experimental work. For the work involved in this thesis, development of the learning algorithms and their interfacing requirements to the existing code was implemented, the remaining framework was previously developed by the CILab group [3].

Operation of the learning methods during target tracking interfaced with several code modules. First, the tracker (either Tracker or LTracker) grabbed an image and then determined the coordinates of the target either through template matching or average grey levels. The coordinates were then passed via the AlgWrapper class to the learning algorithm in the form of a state. At this point the learning algorithms took over and selected an action, developed the reward and updated the action values and policies as required. The action selected by the learning algorithm was then passed back to the tracker in the form of (x,y) coordinates, resulting in a command issued to the camera servos to reposition to the target coordinates if they differed from the previous position.

Each of the three RL algorithms included in this work operated in a similar fashion, however with the addition of the rough feedback variant,

a few differences were encountered. First, at each step in any given episode, the details for the current state, action selected, and corresponding reward were stored in an ethogram-like data table that was written out to a file in plain text. Since episodes varied in size, averaging around 5000 entries, the ethograms were restricted to the first 1000 steps to limit the amount of processing required. Once an episode was completed, rough coverage values for the first 1000 steps were generated and then averaged. The resulting average was used as a performance metric to gauge how well the learning method was performing. The average rough coverage value was then used to adjust the correction step size or the learning rate, α. Besides the periodic revision of the learning rate, the rough coverage variants of each algorithm operated using the same structure and format for tracking, passing data and updating their respective models of the environment.

For verification purposes, the server and imageserver classes provided a means to see the camera's field of view. Again, both of these methods were previously written by the CILab group before development began [3], and they were employed for preliminary testing to monitor how well the tracking methods compared to one another. During the tracking process, when the video feed for the imageserver was in use, the images grabbed by

the tracker were passed to the server and displayed on a separate terminal. This proved useful for verifying correct movements toward the target for any given step and was a helpful troubleshooting tool for the original setup when developing the average grey level tracker. With the tracking and learning aspects of the software setup discussed, the next step included the means to operate the hardware through the PIC controller.

4.4 PIC Control System

The final diagram, Fig. 4.3 outlined control of the motors from the PIC. The serial link connected to the TS-5500 is bidirectional in nature as both controllers were required to communicate information back and forth. The PIC was responsible for controlling the position servos as well as the locomotion drive and each motor had its own custom parameters necessitating individual treatment when designing their respective control methods.

The most prominent constraint that the PIC had to deal with was supporting the position servos. Each of the servo motors required a control signal that needed to be updated periodically or the position would have been lost. The range for the pulse cycle signal was between 12-26mS for updates. This implied that all of the operations for the PIC controller required com-

pletion within a 26mS interval. Each of the different servo motors exhibited different performance specifications when it came to timing and control signals. The pulse width or range of the signals were as follows. The Hitec servo had a range of 1.1 - 1.9mS, the Hobbico servos had a range of 0.7 - 2.06mS and the Futaba servos had a range of 0.363 - 2.06mS. The highest values for the respective range were used when allotting time in the interval for each of the four servos actuating the line crawler. The total time required for the servos was 8.08mS out of a possible 26mS interval.

In addition to controlling the position servos, the locomotion drive was also operated by the PIC. The dc motor control was achieved through the use of an h-bridge IC that required two digital inputs to dictate what operation was required. The values and their corresponding results can be seen in Table 13. The amount of time to place a 0 or a 1 on two output ports was negligible, taking only a few clock cycles for the PIC (operating at 4MHz). The resulting time to control all of the devices was well below the minimum 12mS interval time.

With the addition of a command to report the camera location, an additional 2.08mS were added to the total time necessary during one interval. This allowance was for up to two bytes transmitted at 9600 bits per sec-

ond to the TS-5500 on the serial data link. The total time required for any interval using the maximum values was 10.16mS. During the verification stages, it was discovered that using less than the minimum time for the refresh rate of the control signals for the servos was a bad idea. The servo motors exhibited erratic behaviour that could have caused potential damage to the robot or individuals handling it. As a result, delays were introduced to avoid this problem.

There were two sets of delays used in generating control signals to maintain the update interval length. The standard millisecond delay routine was provided in the CC5X development package. Since there was no support for finer delay times, a set of micro-second delay routines were developed for convenience. They included a 5μS, 100μS, and a variable delay based on an input count passed in to the method. The variable delay was used to generate the bulk of the length of each control signal providing a resolution of 8.08μS for each input count. The other two routines were fixed delays, providing either 5 or 100μS extra time to the main delay generated with the variable method to shore up the control signals as required. The percentage of error for the variable delay was +/-0.25%, which was small enough to be undetectable by servo movements.

To keep the servo control signals updated at regular intervals and allow for any potential problems that could slow down the PIC, an extra delay of $10mS$ was added to the existing $10.16mS$ processing time already in use. This allowed the servos some time to adjust to their new coordinates before polling for new commands and at the same time did not leave that much time between intervals that the servos would lose their positions. Also, polling the UART serial link every $20.16mS$ proved successful for gathering input commands from the FIFO buffer. During regular operation for the TS-5500, commands were never issued in quantities that exceeded the two byte storage capacity before being read and acted on by the PIC controller. The extra delay time also has the potential for future expansion into additional commands or devices to control.

One final thing to note about the PIC servo control was the special consideration provided to the line grip servos. Position control servos move as rapidly as they can toward the target position indicated by their control signal. As a result, stronger servos operating the line grip tended to move the upper and lower chassis modules with considerable speed, having the potential to cause damage to either the line crawling robot or anyone nearby. To prevent this problem, limiting code was implemented to stop the servos

from moving to updated positions with a maximum velocity. Instead, one step was allowed per interval, allowing approximately 50 degrees of movement per second which was more suitable for safety concerns.

Verification of PIC control classes were performed with output devices to provide visual feedback. First, testing the serial link was done by developing a simple test program for linking any terminal to the PIC serial port through a standard communication port. Using a local feedback loop on the PIC, any data sent to the PIC from a terminal was read in from the buffer and sent back out to the origin. This was the simplest means to verify that anything typed at a keyboard was echoed back to the screen indicating that the serial link to and from the PIC operated correctly.

Another simple test program was developed to verify the PIC control of the servo motors. Using the + and - keys as positive and negative steps, the control signal output to a servo from the digital output of port B on the PIC was successfully controlled. This resulted in movement in both the positive and negative directions for all servo motors. A more in depth discussion of how this was used to calibrate the servos and setup control to handle each type of servomotor follows in Sec. 4.4.1. Successful control of each type of servo verified the hardware and software connections from the PIC to

the motors. The dc motor was included in the same test program with four possible commands corresponding to the numbers 1 through 4 (forward, reverse, brake, and stop respectively). The dc motor responded exactly as expected when powered through the TPIC0108B IC and operated by the logic levels from the PIC. After verifying proper operation of each device individually, they were all added together to ensure that the amount of time between update intervals was sufficient to keep the servo positions without faltering. Proving proper operation at this stage confirmed that the design worked as planned.

4.4.1 Calibration of Position Control Servos

This section discusses calibration of the different types of servo motors. Since each motor exhibited slightly different parameters for the control signal, the width of the pulses differed from one servo to the next. Each motor is discussed in turn, providing the number of steps available and the resolution.

First, the Hitec servo responsible for operating the upper line grip was calibrated. The control pulse width varied from 1.1mS - 1.9mS. Using the variable delay routine, delay_var(), for generating the control pulse, the min-

imum resolution available was 8.08μS per step with an error of +/-0.25%. Using this resolution, control of the Hitec servo required 99 steps from 0 to 180 degrees. The resolution for each step was approximately 1.82 degrees. Since this servo was operating the upper line grip, finer resolution was not necessary as coarse control was sufficient to provide the required open and closed positions. The range for the steps started at 136, which corresponded to 1.102mS (approximately 0 degrees), to 235 steps, corresponding to 1.894mS (approximately 180 degrees). Using these step sizes, it was possible to specify the complete range of the Hitec servo with one unsigned 8-bit integer.

Next, the Hobbico servo that was responsible for operating the lower grip as well as the camera tilt action was calibrated. The control pulse width varied from 0.7mS - 2.06mS. The variable delay routine was used again for generating control pulses. With the same resolution of 8.08μS per step, 87 steps corresponded to the zero degree position, at 0.703mS and 255 steps corresponded to 180 degrees, at 2.0604mS. The range for the Hobbico servo was 168 steps, resulting in a resolution of 1.07 degrees per step. For convenience, a single unsigned 8-bit integer was used to specify the position of the Hobbico servos.

Finally, the Futaba servo that was used for operating the camera pan action was calibrated. The control pulse width ranged from 0.363mS to 2.06mS. Using the variable delay routine to develop the control pulses resulted in a range of 45 steps, corresponding to 0.3636mS at the zero degree position and 255 steps, corresponding to 2.0604mS at the 180 degree position. The range for the Futaba servo contained a total of 210 steps, providing a resolution of 0.86 degrees per step. Similarly to the previous two servos, a single unsigned 8-bit integer was used to specify the amount of steps and hence the position of the Futaba servo as well.

After establishing the range of steps to operate each of the different types of servos, it was possible to operate all four servos in a predictable manner resulting in desirable behaviour from the line crawling robot.

4.4.2 Simple Locomotion Tests and Verification of the DC Motor

As an extension to the information provided in Sec. 4.4 with regards to the dc motor and its verification, there were several tests performed to ensure proper operation. Three generations of locomotion drives were tested in the same fashion. First, a very simple functional test was performed, followed by control experiments using the h-bridge and then implementation into the

line grip driving a payload.

The first test was simply to apply power across the motor terminals to ensure it would turn properly. All three vintages of dc motor passed this test easily as it verified the drive being tested was not broken. Next, the intelligent h-bridge IC control was added and control commands were issued through the PIC. Each motor passed this test as well since it was mainly to verify proper circuit operation of the control board. The last test involved installing the locomotion drive into the power train and hauling a full load. Only the drive from MicroMo was sufficiently powerful to pass the final test with the revised, heavier payload. The torque output from the new motor was verified through creating varied angles of inclination on the skywire test setup and forcing the robot to navigate up the inclines(see Sec. 5.1).

4.5 Robot Behaviour

The behaviour of the line crawling robot followed the hierarchy setup in the discussion in Sec. 3.12. From the point of view for the software systems, the behaviour was kept as simple as possible to reduce the processing demands on the TS-5500. Once the line crawler was released on the sky wire, a start command was issued via triggering any of the contact switches. At that

point the general behaviour was to move along in the direction of travel corresponding to the side that the switch was pressed. During its travels, the sensors were active and being monitored, including the contact switches, the infrared sensors and the camera.

The software implementation of the behaviour rated the contact switches as the primary source of input for detecting obstacles. The contact switches were connected to digital I/O (DIO1) onboard the TS-5500. The voltage levels registered either 0 or 5 volts for closed and open circuit operation respectively. When the contact switches were closed, the pin on the DIO header was driven low and the level 0 behaviour for obstacle avoidance or evasion took over. At that point, a stop command was issued to the dc motor to avoid any further contact with the obstacle. The next step was to reverse away from the obstacle and then use the camera to sweep for targets of interest.

The infrared sensors were continually monitored using the analog to digital converter on the TS-5500. The input levels provided enough information to make the TS-5500 aware of the status of the grip closure whether it was closed or open and potentially obstructed. In the event that a problem was encountered, the dc motor was shut down and re-positioning of the line grip

servos took place. At this stage of development if a problem still existed, operator assistance was necessary to resolve the issue.

The final sensor implemented in directing the behaviour of the line crawling robot was the camera. This was a second layer device, implying that it had the power to sub-sume control of the robot to obtain its goal. The goal of the camera was to acquire salient images of targets of interest such as power line insulators. In its current capacity the camera was used for data acquisition only but plans were in place to extend its functionality to scan for targets during regular operation of the line crawler. For the event when an obstacle was spotted by the camera, commands would be issued to the locomotion drive to stop so that the target could be centred in the field of view to acquire an image. At that point, control would resume to the lower layer and movement would continue.

4.5.1 Sensor Verification and Calibration

This section covers verification and calibration of the contact switches and the IR sensors for the line grip. A couple of simple tests were performed to determine suitability and ease of interface with the TS-5500.

First, all four contact switches were wired to DIO1_0 - 3 on the TS-

5500. A simple test program was written to monitor the digital I/O pins. The contacts were considered normally open, as were the contact switches. This resulted in a logic high voltage on the DIO pins until a switch was closed, driving the pin to a logic low level. To verify proper operation, a visual representation was programmed so that each pin was monitored during operation and a one was displayed if the contact was open, otherwise a 0 was displayed indicating a switch was closed. All voltage levels were verified with a voltmeter, confirming that the four switches were operating properly, both individually and simultaneously.

Next, the IR sensors were calibrated and tested in circuit to verify performance. The first experiment was to test the IR sensors and see how they responded at varying distances in both straight and lateral directions. The components used for this experiment were Fairchild Semiconductor, QRD1114 Infrared Reflective Optosensors [13]. The experiment was split into two parts, first was a straight distance comparison (transmitters and receivers pointing directly at each other) with the value of the detectors using an 8-bit analog port on the controller. The second experiment was done by moving the sensors laterally away from one another and comparing the same 8-bit analog value. The results of these experiments were used to ver-

Straight Line Distance	Analog Value Sensor 1	Analog Value Sensor 2
$0.6cm$	7	9
$1.0cm$	7	9
$2.0cm$	9	11
$3.0cm$	11	13

Table 17: Straight distance vs. 8-bit analog count for IR sensors

ify the suitability of adding IR sensors to the line grip for monitoring proper closure.

The first experiment compared the 8-bit analog values versus separation distance of the two sensors. Two transmitter-receiver pairs powered from the regulated 5-volt power supply were used. The resulting data shown in Table 17 demonstrated the trend for straight line distances and their corresponding analog values. The trend shown in Table 17 is very slow to change with the straight line distance. The reason that readings were stopped after 3cm is that the maximum straight-line distance that is possible with the line grip was less than 3cm, after that point the upper and lower grip separate far enough that contact would no longer be possible. The data exhibited an appealing trend where the values changed very little when a straight line of view was available from one sensor to the other at short range.

The second experiment involved measuring lateral distances and comparing the resulting 8-bit analog values at each distance of separation for the

two IR sensors. A similar setup as the first trial was used with straight line distances (see Fig. 4.4). The straight line distance between the two sensors

Figure 4.4: Setup for the IR sensor experiments

was kept constant at 2cm, comparable to the actual distance of the sensors when both upper and lower grips close over the skywire, whilst the lateral distance was varied. Table 18 shows the results of the lateral distance versus analog value experiment. The results of the second experiment showed that when the IR sensors were separated by a short distance in a horizontal direction (out of line of sight) the analog values ramped up quickly to the

Lateral Line Distance	Analog Value Sensor 1	Analog Value Sensor 2
$0.5cm$	91	11
$1.0cm$	139	12
$1.5cm$	172	51
$2.0cm$	193	101
$2.5cm$	235	253
$3.0cm$	255	255

Table 18: Lateral distance vs. 8-bit analog count for IR sensors

maximum value at around 3cm distance. This was a very promising result since the intent of placing IR sensors on the line grip was to monitor for conditions when the line grip had closed properly and the sensors were within line of sight and then the opposite condition when an obstruction was preventing the grip from closing properly. Since the grip opens in an arc, the IR sensor values rapidly drop away as the grip opens, signalling a problem has occurred or that the grip is open.

The results provided in the tables proved that the IR sensors were a suitable match for the desired task of monitoring the grip to ensure proper closure has occurred. The sensors are excellent for monitoring straight-line distances and since the grip separates rapidly after opening there would be no mistaking a grip malfunction by accident.

4.6 Optimization

There were a couple of areas investigated for optimization during development. The timing interval for the servo motors was a concern, so some effort was put into attempting to reduce the amount of time necessary to send out control signals and allow for more future expansion or to free up processing power. Also, the amount of time and processing power used during the reinforcement learning methods using rough coverage was another focus for optimization.

Initially, it was thought that the PIC would poll the serial port as often as possible to detect when command bytes arrived in the buffer. The intent of checking the buffer frequently was to avoid a loss of command data in the event that the bytes were issued too quickly as there was only storage space for two. At a later stage of development it was decided that issuing commands more frequently than two in the span of one $20mS$ interval would be very unlikely and with feedback added from the PIC, corrective actions could be taken from the TS-5500. However, reducing the amount of time that the PIC issued control signals was still useful to the line crawler as it allowed further expansion to more devices and streamlined operation. The

means by which the timing interval was reduced pertained to the operational envelope of each servo motor. Each position servo had a range of approximately 180 degrees, with a neutral position being approximately halfway. This corresponded to the length of their respective control signals as well. This meant that if the servo range was considered from -90 to +90 degrees, the shortest possible control signal would be at -90 degrees. The line grip servos operated with a 90 degree envelope, positioning the starting point to the lowest possible value, leaving enough rotation to satisfy the 90 degree minimum yielded a significant time savings. The upper line grip normally remained closed during operation, this was positioned to coincide with the -90 degree set point, allowing a rotation of up to 180 degrees from the closed position. This didn't interfere with the work envelope for the upper line grip as it rotated up and never closed any lower than the driving position on the sky wire. As a result, the maximum control signal length was reduced to $1.5mS$. The same principal was applied to the lower grip and the camera tilt servos, positioning the motors so that the 90 degree range of motion was available, starting with 0 degrees at the neutral position when grasping the sky wire and positioning the camera respectively and then moving to -90 degrees when fully open or tilted straight down. This reduced the maximum

control signal length by a similar amount, to 1.52mS each, for a total reduction of 1.48mS. Although this was only a small time savings, it was significant enough to provide room for one more control signal and helped reduce the amount of time that the control lines were active allowing further expansion of devices or commands.

The other area for optimization that was investigated related to RL methods during the target tracking process. Preliminary testing demonstrated how computationally intensive the learning methods were when the rough coverage methods were operating. Each time an episode completed and the new coverage values were generated, the processing requirements were so expensive that the camera system stopped tracking for a short period of time. This was problematic as it was important to keep the target in the field of view at all times to acquire the best images. As a result, a means to limit the computational requirements of the rough coverage learning methods was investigated. As previously mentioned, the number of steps considered out of each episode were reduced. Originally, all steps were gathered together and included in one large ethogram for generating rough coverage values for each behaviour. To limit the amount of processing necessary, approximately 20% of the steps in any given episode were used from the ethogram

to generate the coverage values. This provided a large enough sample of any episode to give a reasonable view of the behaviour performance and at the same time yielded a significant reduction in computational power, minimizing the delay during the tracking procedure.

5 Experiment Design

After verification of the system components, the next step was to prove the original conjectures through experimental work. Test situations were developed through an iterative design procedure for both the hardware and software systems to provide a collection of results from which conclusions could be drawn regarding system performance.

5.1 Hardware Experimental Environment Setup

In order to run some of the experiments, a testing environment was necessary to provide the required situations. The first aspect in setting up a testbed for the line crawling robot was to develop a prototype tower structure. Samples of line clamps, and vibration dampers were provided by Manitoba Hydro (see Fig. 3.5). In addition, they provided approximately 10 metres of rolled skywire and drawings outlining the dimensions of actual tower structures for the lines along Bishop Grandin Boulevard in Winnipeg. With all of the available parts and knowledge, a prototype testbed setup was built.

The tower setup was constructed with the intent of re-creating similar constraints that the robot would experience on the skywire but in much safer

conditions for experimental purposes. To provide a similar habitat for the robot, one tower needed to be designed to support both types of obstacles from Fig. 3.5, including 2 vibration dampers and one line clamp. Two additional virtual towers were required to support the skywire and allow for adjustment to the slope of the line between towers. The line crawler needed to be able to navigate up and down sloped sections of skywire. The end towers were used solely for the purpose of anchoring the skywire and adjusting the slope of the wire. As a result, simple posts with a secure base and attachments to hold the skywire were sufficient. The central tower containing all of the obstacles required a more complex design to yield a useful experimental simulation tool.

To keep the dimensions of the prototype tower within reason, they were halved. Since a half-scale design was used for the dimensions, only half of the space that is actually available in the field will be used, thus guaranteeing that success in the testing environment would extend to field trials concerning space constraints. In addition to the dimensions, the other design consideration observed for the prototype towers was the weight it needed to support. The towers were expected to manage the weight of two vibration dampers, one line clamp, the sky-wire and the line crawling robot. A to-

tal estimated weight of 15kg implied that the tower structures needed to be fairly sturdy. Reinforcing the trapezoidal structure of the tower was done through adding extra 2x4's along each side of the trapezoid. Grooves were also cut in the top and bottom of the structure to mate with the support beams which were cut both top and bottom at 19 degree angles for slotting into the grooves. This can be seen in the construction drawing (see Fig. 5.1), which displays the shape, measurements and materials used. Unlike the wooden

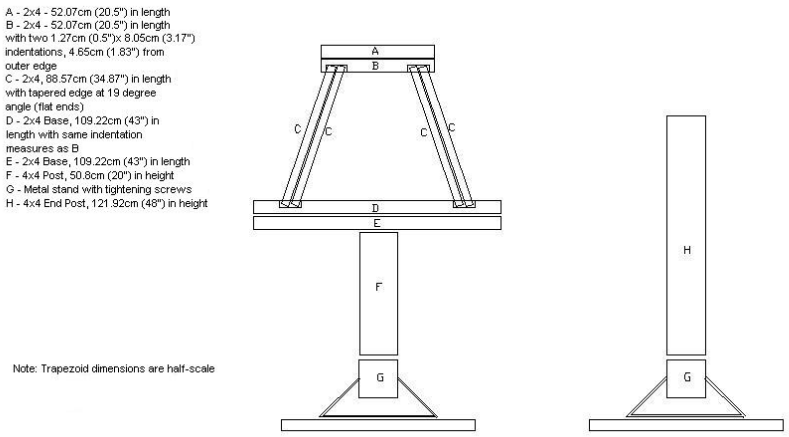


Figure 5.1: Construction diagram for prototype tower

structure, the base of the platform was a pre-constructed steel stand with a wide stance intended for supporting a large Christmas tree. The completed tower can be seen in Fig. 5.2, including the steel base support structure.

The central tower contained several modifications to improve its strength.

Figure 5.2: Completed prototype tower

In addition to strengthening the frame, selection of appropriate fastening devices was made with a similar intent. Three inch wood screws and 2.5 inch spiral nails were used to fasten the pieces of the trapezoid together. Since the line clamps had custom fittings for attaching to hydro towers, it was necessary to come up with a modified attachment to connect the line clamp to the prototype tower. The image in Fig. 3.5 shows all of the obstacles but also the right most line clamp has a foam-covered steel hook threaded through it that was subsequently screwed into the underside of the top of the trapezoid, providing a support for the skywire. The end towers consisted of standalone posts that rested on similar stands compared to the central tower. Instead of supporting all of the obstacles, they were required for fastening the ends of the sky wire to maintain some degree of rigidity allowing the line crawler to travel back and forth and providing control over the slope of the skywire (see Fig. 5.3).

Completion of the prototype towers provided a safe alternative test environment for simulating line crawler operations. This eliminated the need to use actual power lines to perform basic verification and formal experiments, resulting in a safer environment for both the robot and people involved.

To facilitate the experimental work with the target tracking system, a

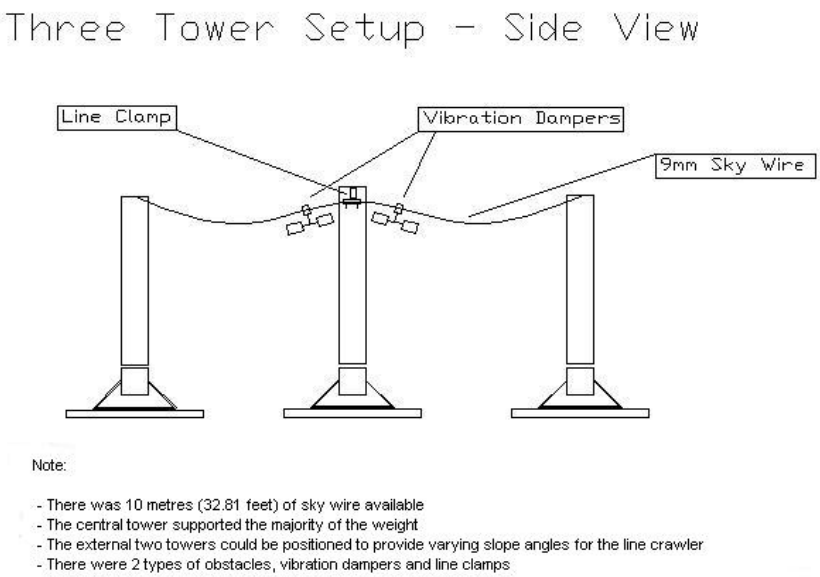


Figure 5.3: Construction drawing of complete prototype tower setup

replica of the monocular vision system was built separate from the line crawling robot. The intent of having a separate system for testing and experimental work was to eliminate the need for battery power, periodic recharging and to reduce any possible risk of damage to the line crawling robot. The target tracking experiments required large periods of time to gather useful data and building a safe testing environment was very desirable to avoid the need for constant supervision.

The replica of the vision system was built to provide performance as close to that of the line crawling robot as possible. The same combination

of two servo motors, one for the panning action and one for the tilting action were used. A flat surface comprised of the previous platform material made from aluminum provided the base to operate from. The camera and its servos were connected upside down, corresponding to the same orientation and position of the motors on the line crawler as the experimental platform was inverted. The resulting experimental monocular vision system can be seen in Fig. 5.4. As the servos were connected directly to the platform and cam-

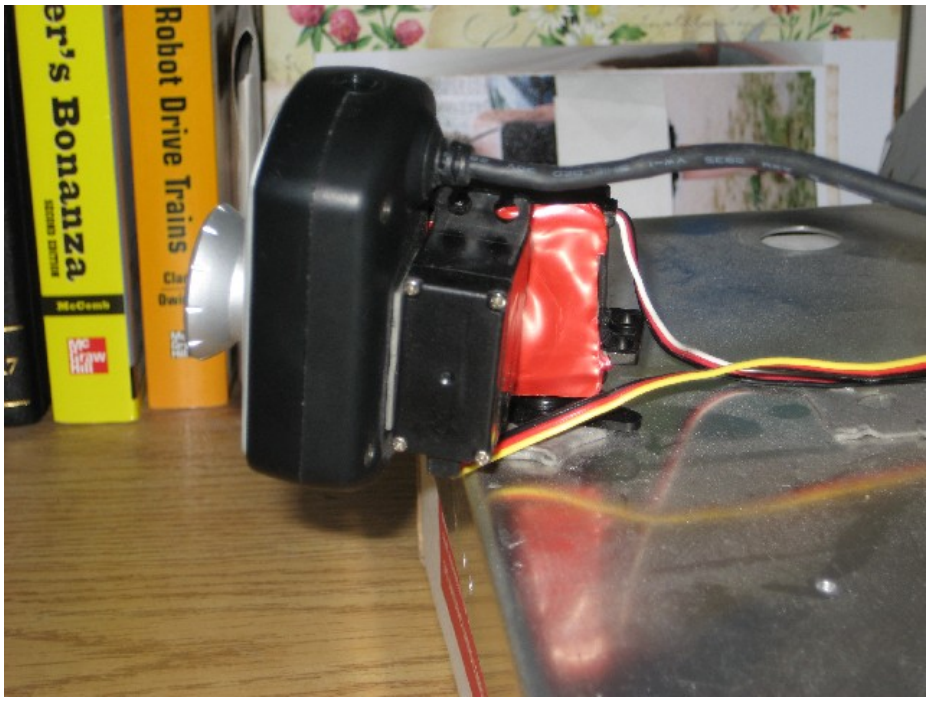


Figure 5.4: Replica of the monocular vision system from the line crawling robot

era respectively, the same work envelope was available for movement. The

resulting test system was built as close as possible to the existing system on the line crawler.

5.2 Hardware Controllable Parameters

The experimental test vector parameters included several variables that were considered. During the target tracking experiments, the two variable parameters included were distance from the target to the camera and the height of the target. An initial target for the line crawling robot was to capture images of insulator stacks. After examining actual dimensions of towers relating the location of the skywire in comparison to the insulators, the distance versus height ratio was approximately 4:1. Maintaining this same ratio, the distance from the target during tracking experiments was selected to be $17cm$, and the height of the target corresponded to $4.25cm$. These parameters were monitored and maintained throughout the experimental work. The other adjustable parameter was for locomotion tests, requiring varied angles of inclination and declination of the skywire to monitor performance of the locomotion drive. These three parameters comprised the variable elements from a hardware perspective during the experimental phase.

5.3 Software System Parameters

There were a number of different variable parameters in the software systems. A discussion related to each RL algorithm and its associated parameters are included for the actor critic, Q-learning and sarsa methods.

5.3.1 The Actor Critic Algorithm

The Actor-Critic algorithm contained several variables (see Sec. 2.3.1). Selection of the initial values for alpha, beta, gamma, the preference values and the policy π were made as follows. The learning rate, α was chosen to have a value of 0.1, preventing any large correcting steps during the learning process. Although this limited the learning process from taking large steps in a corrective manner, it also prevented large steps in the wrong direction, preventing overshoot from getting out of hand. The step size parameter, β for controlling the preference adjustment was also conservatively selected at a value of 0.1 to avoid giving too much or too little preference to actions that performed well or poor early on and mistakenly either avoiding or over selecting them as a result. The discount factor, γ was initialized at a similar low value of 0.1 to avoid giving too much weight to future states, making the system somewhat near-sighted. The reason for choosing a value

of γ so low was due to prior knowledge of the environment and the minimal amount of noise expected. As a result, current experiences would not differ much from longer term behaviour, so weighting future states more heavily was unnecessary. A parameter, ϵ which prevented the policy π from always being greedy was chosen as 0.1 as well, implying that 10% of the time, exploratory actions were taken and the rest of the time, deterministic actions were taken. The preference values were all initialized to zero which in turn setup the starting values for the policy, π. The initial policy values were unbiased by setting all values to 1, this meant that it was equally likely to select any state before experience was gathered. Besides the parameters discussed, the number of states and actions were pre-determined as discussed in Sec. 3.11.1.

5.3.2 The Q-Learning Algorithm

As opposed to the actor-critic method previously discussed, Q-learning concentrated on learning action values to generate a policy rather than having a separate policy and value function. As a result, there were some different initial parameters, but to keep their respective performances on a reasonably level playing field, similar parameter values were chosen when possible.

The values of α, ϵ and γ were once again selected at 0.1. The action values, Q(s,a) were initialized by setting all values to zero and the initial policy, π was set to an arbitrary soft or non-greedy policy with at least some chance to visit all possible state-action pairs. As before, the number of states and actions were fixed to match the specifications provided in Sec. 3.11.1.

5.3.3 The Sarsa Algorithm

The sarsa algorithm was included to provide a comparison for a value function learning algorithm that followed the policy it searched with. In essence, it was the counterpart for Q-learning, providing an on-policy approach for comparison. The parameters were chosen with the same values for α, ϵ and γ at a value of 0.1. The action values, Q(s,a) were again initialized to zero, and the policy, π was selected as an arbitrary soft policy as well to provide a similar environment for comparison during experimental work.

5.4 Test Vector Development

Generating a set of test vectors that were designed to exploit the the differences between the various learning methods employed was an iterative process. Careful selection of components including the period of time for

each experiment, the distance and height of the target, movement patterns and the addition of noise were all considered.

The period of time for each of the RL tracking experiments were synchronized for all algorithms. Various amounts of time for the learning process were provided to allow for slower learning or those methods that required more experience. Times ranged from 1 minute up to 2 hours, with intervals in between of 5 minutes, 15 minutes, and 1 hour lengths. The distance and height of the target tracked was selected and fixed (as discussed in Sec. 5.2) matching the distance ratio of actual insulators related to their position and the skywire. The movement patterns of the target exhibited two patterns, circular motion and random trajectories, both were included to discover if the learning methods may have incurred an advantage from a predictable movement pattern. Finally, noise was added to provide some degree of similarity to environmental stresses that were possible outdoors, including high winds, uneven lighting, or precipitation to name a few. Noise was added in the form of distorting the coordinates of the target location. Essentially, fooling the line crawling robot into thinking the target was located somewhere else in its field of view other than the true location.

The controlled additive noise was limited to the dimensions of the field

of view. Implying that the mis-perceived location still remained within the field of view of the camera, but it was incorrect compared to the location of the actual target. Generating the additive noise was accomplished through the use of a pseudo-random number generator to supply uniformly distributed random numbers ranging from -1 to 1. The random values were scaled up to integer values and added to the target location to attempt to confuse the tracking system. The random number generator was considered pseudo-random as it relied on a seeded value from the system clock to generate a sequence of random numbers [51]. A uniform deviate was selected to provide an equally likely chance of any value occurring throughout the range of -1 to +1.

The last adjustable parameter in the test vectors was the speed of movement for the object being tracked. The preliminary experimental work was performed with a fixed speed of (6,6) which refers to the amount of pixels moved in both the x and y-directions in 2 dimensional space that the camera can see. The time between movements was $50mS$, resulting in a total movement rate of (120,120) pixels/second. The velocity of the line crawling robot was insufficient to warrant testing much higher speeds but additional test vectors were constructed containing higher rates of travel for the obsta-

cle to discover the effects of obstacle speed on performance. The range of speeds were from (6,6) to (10,10) pixels per $50mS$.

5.5 Line Crawling Experiments

At the time of writing this report, only preliminary experimental work had been done with the line crawling aspect of the robot. There were a number of areas involved that slowed down the final construction and as a result, preliminary testing was the only stage completed. Simple locomotion tests to prove that the line crawler was self sufficient and could wander back and forth on the sky were performed as well as angle of inclination tests to see how steep a slope was traversable in the current configuration. As such, the system parameters consisted mainly of simple behaviours and responses to sensory input as well as the angle of inclination of the sky wire. The sensory input parameters consisted of what to do upon contact with an obstacle. For example, the instant reaction was to stop the locomotion drive and reverse for a set period of time before scanning for targets of interest to acquire images of. Afterwards, direction of travel was reversed and the same tactics were employed travelling along the sky wire in the opposite direction. For these experiments, the vision system was mounted with the origin facing

sideways off the platform to allow examination along both directions instead of limiting it to either forward or reverse views alone.

6 Experimental Results and Discussion

This section contains the results compiled from experiments with the line crawling robot. They were designed to explore the differences in tracking techniques under a variety of conditions with the intent of acquiring the best quality images and to explore the functionality of the line crawling robot to provide some insight into its capabilities and potential for future expansion.

6.1 Experimental Results

Each aspect of the line crawler that was tested includes a discussion for what was expected, actual results and the associated implications. The main focus for experimental work on the target tracking system included a comparison between the RL methods previously discussed and the classical tracking algorithms, including both template matching and the average grey level method. Several experiments were processed to determine the performance of each algorithm, including varying the time available for the learning process, altering target speed, noise susceptibility, and variable target trajectories.

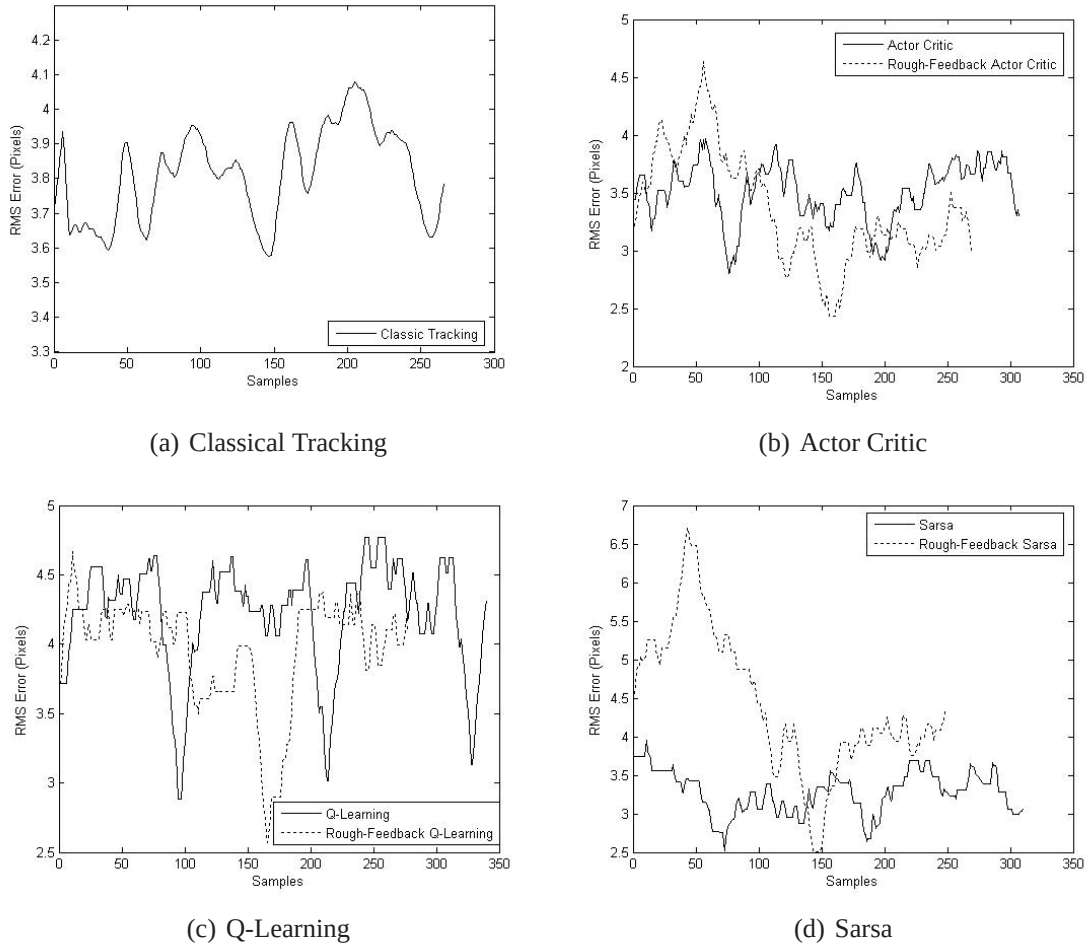
6.1.1 Varied Time Target Tracking

The first experiment varied the length of continuous time for the target tracking process. As the time allowed increased, the results were expected to favour the learning methods as they tend to converge toward an improved policy over time. However, since a relatively static environment was used, the classical tracking approach was expected to fare reasonably well. The differences between the average grey level and template matching methods were also another focus of the time varying trials. Since the number of computations required for the average grey level method were fewer, a greater number of samples were expected along with higher rates of error compared to template matching.

Each tracking method and algorithm were implemented and reported using three different lengths of time, 1 minute, 5 minutes and 15 minutes total. The algorithms included were the classical tracking, actor-critic, Q-learning, sarsa and the rough feedback modified versions of each of the RL methods yielding seven in total for each experiment. This was implemented for both template matching and the average grey level methods, providing fourteen comparisons for each test vector. The variables in the test vectors consisted

of the algorithm in use, the tracking method and the length of tracking time. The static components of the test vectors for this experiment included target speed ((6,6) pixels of movement per 50mS), target distance from the camera (17cm), target height (4.25cm) and target movement pattern (a circular clockwise direction).

The first set of results included were for one minute in duration. The template matching algorithms are presented first and the average grey level methods are presented next in the same format. The first set of 1-minute experiments seen in Fig. 6.1 are somewhat erratic due to the short duration of the test. The learning methods often take a little bit of time to settle after preliminary exploration of new state space. The classical tracking method remains relatively stable throughout as expected since no learning took place. The actor critic and sarsa methods exhibit the lowest RMS error when tracking the target, followed closely by the classical tracking method. The Q-learning method exhibited the most erratic profile and averaged the highest error rates during the first minute. The rough-feedback methods for both the actor critic and Q-learning algorithms provided improved results over time when compared to the classical implementations. The improvements were small, approximately 0.1 to 0.2 pixels more accurate. For such



(a) Classical Tracking

(b) Actor Critic

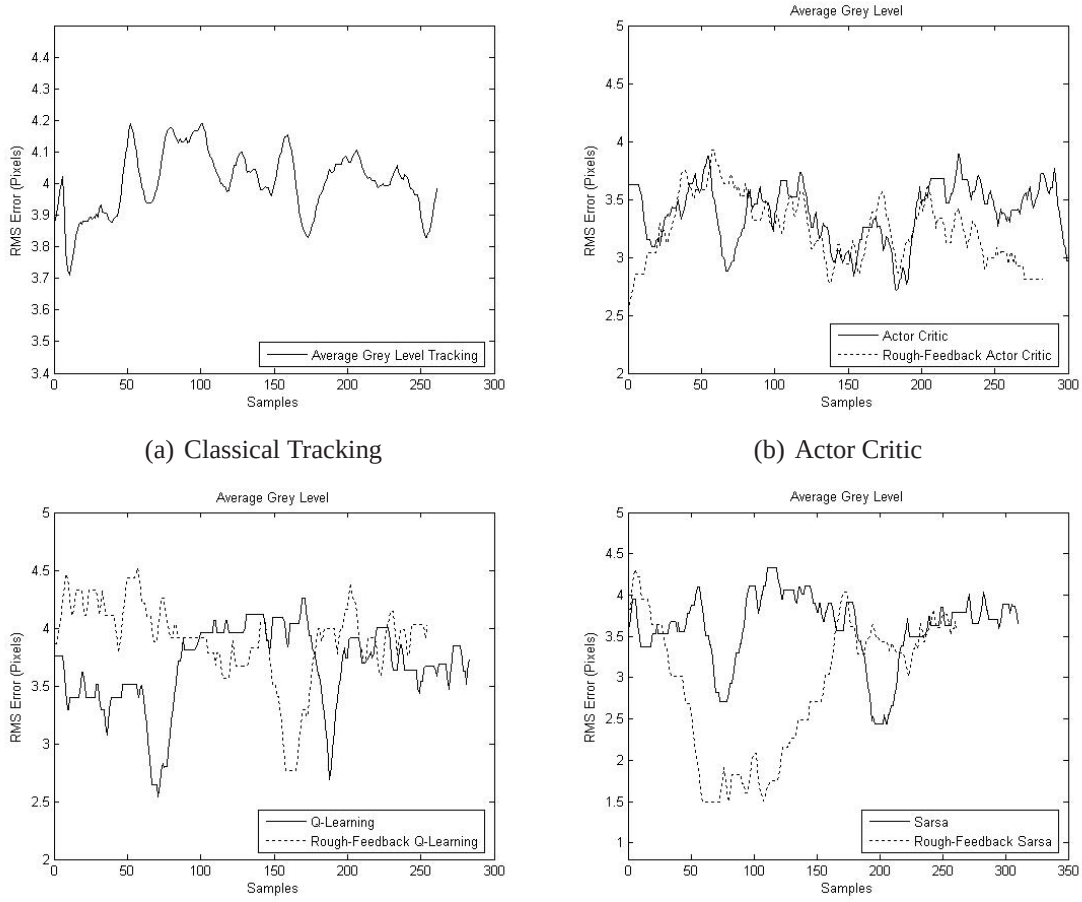
(c) Q-Learning

(d) Sarsa

Figure 6.1: Template matching, 1-minute tracking experiment average RMS error results

a short period of time, this was a promising result indicating that greater lengths may prove beneficial for rough-feedback tracking methods.

The next diagram, Fig. 6.2 demonstrated the average grey level counterpart for the 1-minute tests. The general trend found in the average grey level



(a) Classical Tracking

(b) Actor Critic

(c) Q-Learning

(d) Sarsa

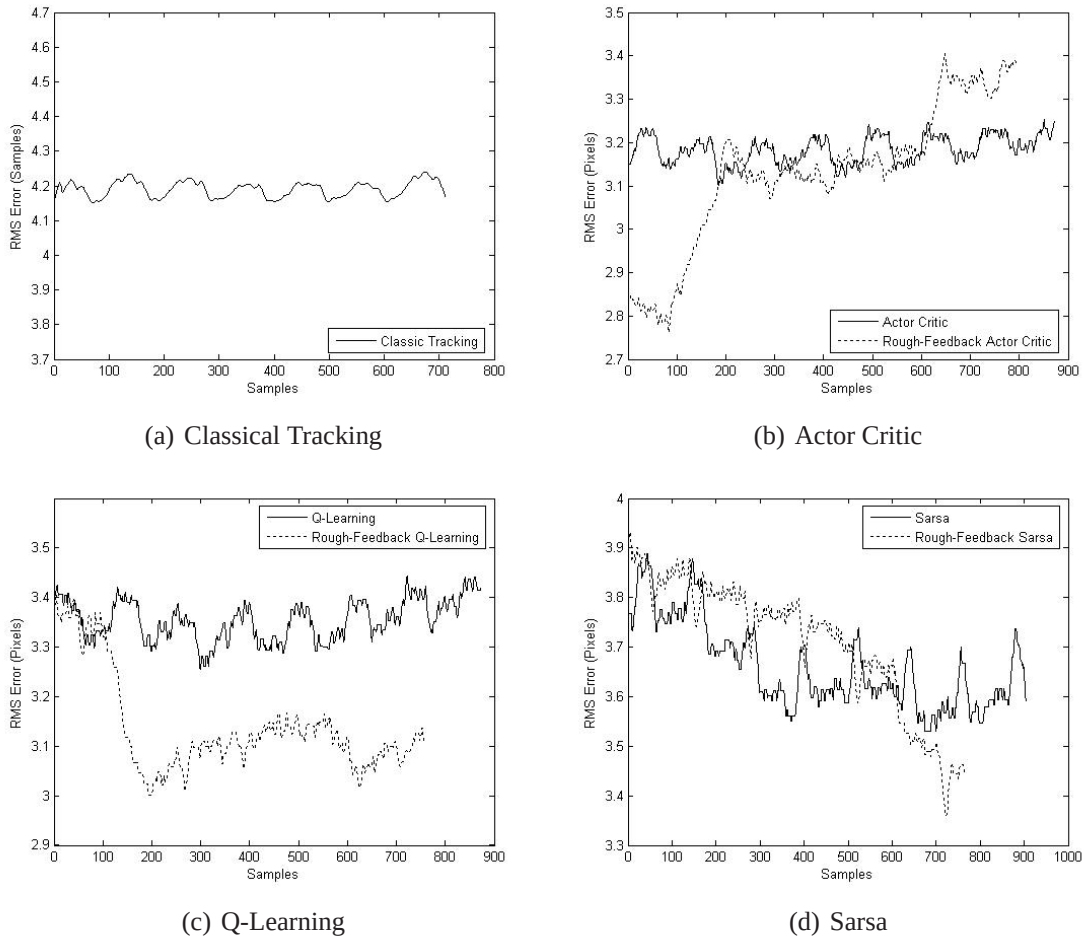
Figure 6.2: Average grey level, 1-minute tracking experiment average RMS error results

method was increased average RMS error and extra samples in some cases.

Greater RMS error can be seen easily in the classical tracking algorithm, averaging 4 pixels compared to the average of 3.8 pixels using template matching. The learning methods were more resilient and provided nearly the same performance. With the exception of Q-learning which performed slightly better, whilst its rough-feedback counterpart did worse. The short duration of this experiment made for potentially erratic results that could change depending upon the initialization and location of the target when tracking began. These problems were faced by both the average grey level and template matching methods, demonstrating the capabilities of each in short duration situations where targets needed to be acquire quickly.

The next set of results maintained the same test vector parameters but with an increased duration of 5 minutes. Both trackers were expected to perform similarly to the previous results but with potential improvement in the RL methods due to increased time to experience the tracking problem and generate an improved policy. The rough-feedback methods were expected to provide performance equal to or better than the classic RL algorithm implementations as the adaptable learning rate allowed for a more intelligent policy adjustment, improving the performance at a cost of extra processing for short durations which would likely result in fewer samples

over the same time period. The results seen in Fig. 6.3 show that several of



(a) Classical Tracking (b) Actor Critic

(c) Q-Learning (d) Sarsa

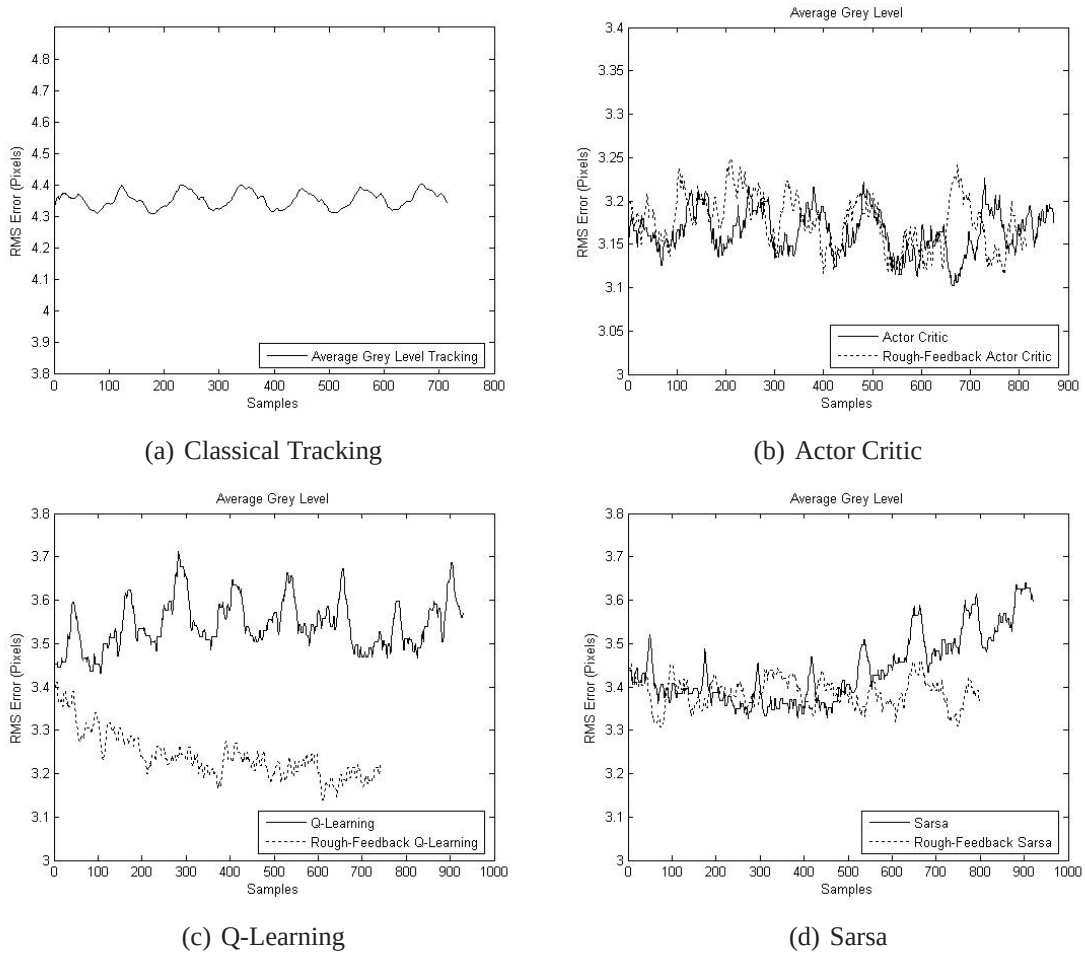
Figure 6.3: Template matching, 5-minute tracking experiment average RMS error results

the performance predictions were accurate. However, the classical tracking method performed worse than in the 1-minute trials, yielding an error rate of 4.2 pixels, most likely caused by varied illumination. Unfortunately the experiments were not all performed at the same time of day and the ambient

light was difficult to control. However, this parallels actual conditions more appropriately as outdoor lighting varies dramatically through the course of a day and based on location. Both Q-learning and sarsa exhibited improved performance with the rough-feedback versions, this is likely in part due to the extra time for learning as well as the adjustable learning rate dependent upon performance. Rough-feedback Q-learning exhibited the least error during the tracking task, averaging close to 3 pixels, closely followed by the classical actor critic method, averaging 3.2 pixels of RMS error. The rough-feedback version of the actor critic algorithm unexpectedly yielded poorer performance over time. This could have been attributed to a host of environmental effects, or perhaps poor suitability for the task. Generating the average rough coverage values associated with the rough-feedback methods required extra time whilst the target was still mobile, implying that it may no longer have been centred or worst case completely out of the field of view. The other results for Q-learning and sarsa suggest that the rough-feedback algorithms were not demanding too much time of the system and perhaps an environmental anomaly was responsible for the actor critic results.

Average grey level tracking experiments of 5 minute duration are in-

cluded in Fig. 6.4. As with the previous 1-minute experiments, greater error



(a) Classical Tracking

(b) Actor Critic

(c) Q-Learning

(d) Sarsa

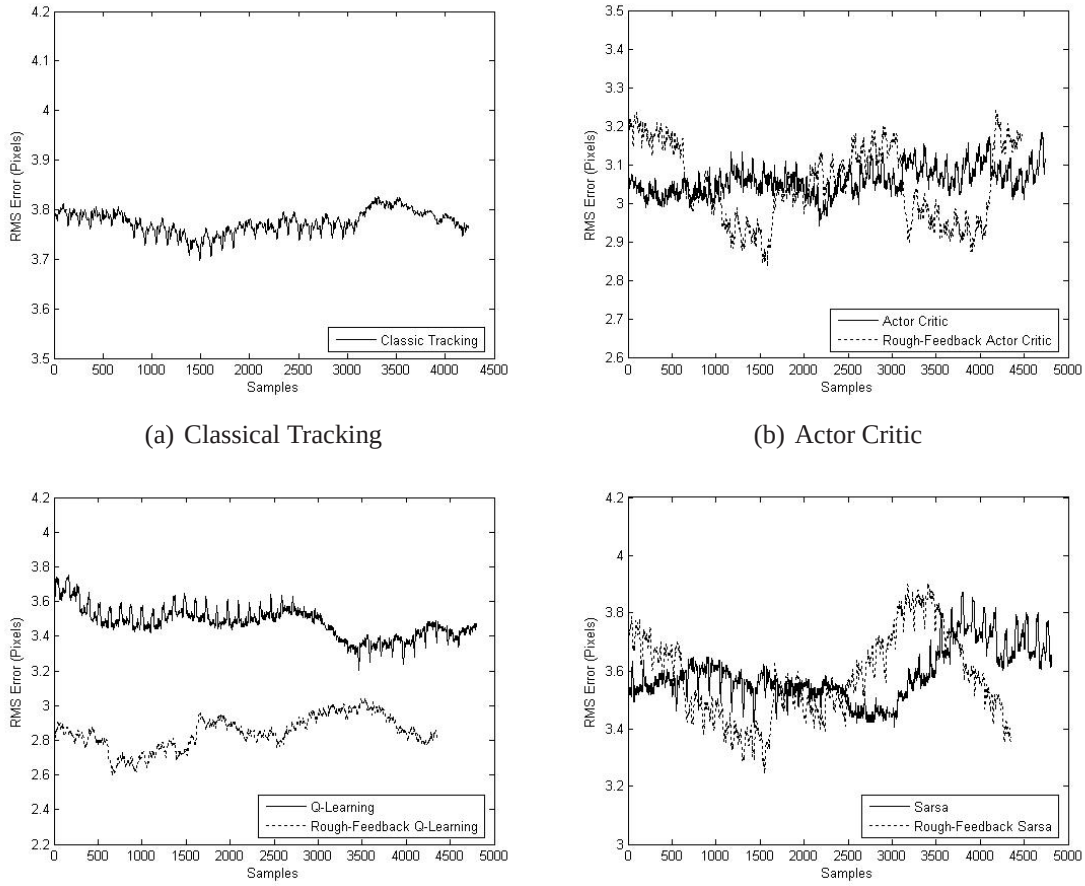
Figure 6.4: Average grey level, 5-minute tracking experiment average RMS error results

rates were expected from the average grey level method, with the possibility of an increased number of samples due to the reduced computational requirements of the algorithm. Aside from that, the remaining performance trends expected in the template matching version of the 5-minute experi-

ment were expected to be re-iterated in Fig. 6.4. The results confirmed the larger error encountered from the loss of resolution of the average grey level tracking method. The classical tracking algorithm yielded an error of 4.35 pixels compared to 4.2 pixels using template matching, approximately 4% worse accuracy. The number of samples were not as different between the two methods as expected with only 10 extra samples provided from the average grey level method (721 compared to 711). The actor critic method demonstrated almost identical performance compared to template matching, with the exception that the rough-feedback version of the algorithm maintained a steady average instead of degenerating in performance over time. The overall error rate was near 3.2 pixels for both algorithms, as before, when using template matching. Both Q-learning and sarsa exhibited greater rates of error over time compared to the template matching experiments. The Q-learning results showed an average increase of 0.2 pixels of average RMS error, corresponding to approximately 6% worse accuracy, whilst the rough-feedback version of the algorithm showed less sensitivity to the tracking method having only 0.1 pixels of increased error, or 3%. The sarsa method exhibited an unusual trend, starting out with a lower average RMS error and increasing over time. The template matching method per-

formed the opposite, improving through learning over time to improved accuracy. The cause of this problem can likely be attributed to environmental factors again as the only difference constituted the tracking method. Since the rough-feedback version of the algorithm performed reasonably well, averaging just under 3.4 pixels of average RMS error, the problem with the classical implementation is likely due to environmental conditions. The accuracy of the average grey level version of rough-feedback sarsa is comparable to the best performance of the template matching method, unlike the classic algorithm, and the Q-learning implementations which demonstrated worse RMS errors. At this point, the results were a bit of a mixed bag and lengthier experiments, comprising of 15 minutes were included to find out if further settling time was required to discover any long term trends.

Next, experimental conditions were kept the same except for the time duration which was extended to 15 minutes to allow for further settling of learning methods to discover long term performance trends. The results for template matching tracking can be seen in Fig. 6.5. The classical tracking algorithm maintained a relatively constant average of 3.8 pixels of average RMS error, as no learning took place, this was expected and mainly used as a base comparison for the remaining algorithm performances. The actor



(a) Classical Tracking

(b) Actor Critic

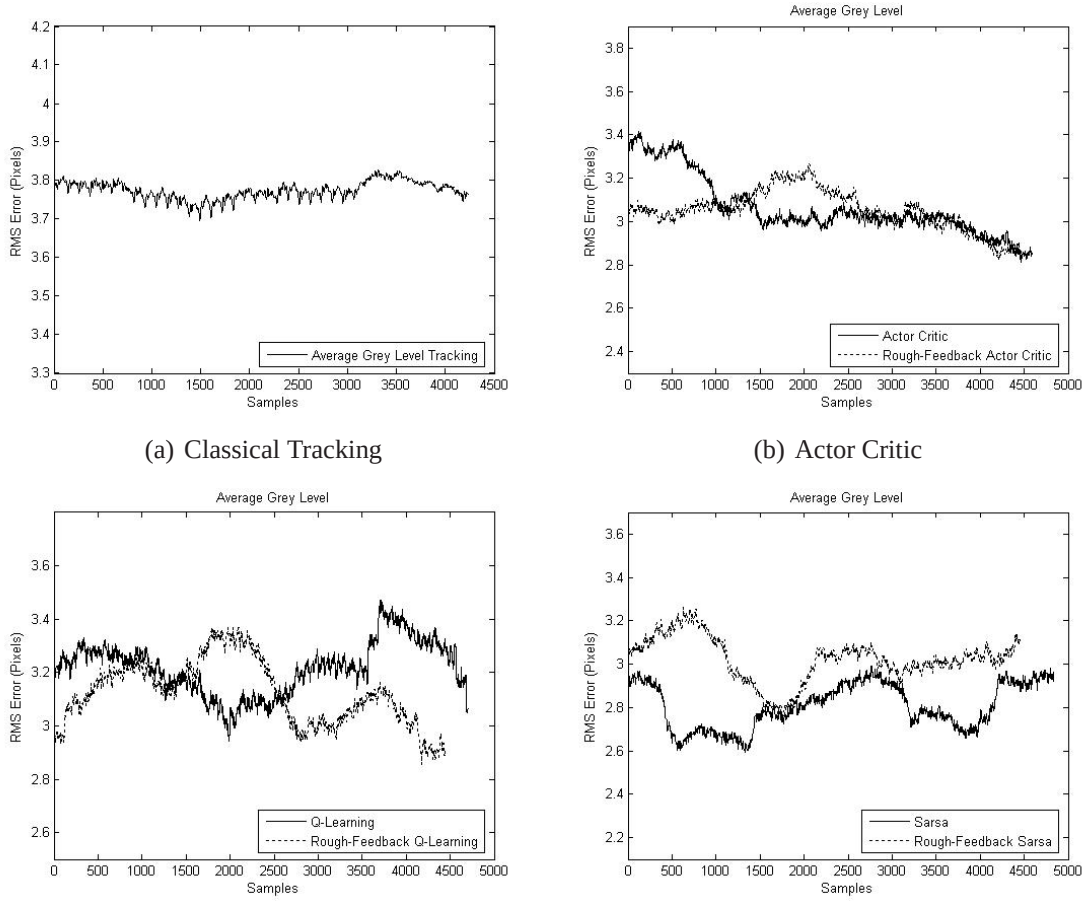
(c) Q-Learning

(d) Sarsa

Figure 6.5: Template matching, 15-minute tracking experiment average RMS error results

critic method exhibited improved performance on average over the longer duration experiment. The average error rate corresponded to 3.08 pixels for the classic implementation and 3.07 for the rough-feedback version. The greatest difference between the two algorithms occurred in the number of samples, the classical actor critic implementation included 300 additional samples that corresponded to time spent generating the rough coverage values for adjusting the learning rate of the modified version of the algorithm. The Q-learning method yielded an average RMS error of approximately 3.5 pixels, an improvement over the classical tracking algorithm, but performing worse than the actor critic algorithm. The addition of rough-feedback Q-learning provided the best accuracy at an average RMS error of approximately 2.9 pixels. Finally, both classical sarsa and the rough-feedback sarsa algorithms achieved an RMS error of just over 3.6 pixels making it the worst performer out of the learning algorithms. In all cases, the rough-feedback versions of each algorithm generated approximately 300 less samples based on the extra computational requirements.

The final experiment in the time variable block was average grey level tracking for 15 minutes. The results can be seen in Fig. 6.6, for each of the algorithms. The classical tracking algorithm confirmed the expectation



(a) Classical Tracking

(b) Actor Critic

(c) Q-Learning

(d) Sarsa

Figure 6.6: Average grey level, 15-minute tracking experiment average RMS error results

of slightly worse accuracy of approximately 3.85 pixels compared to 3.8 from the template matching approach. The number of samples differed only slightly between the local threshold and classical tracking. The actor critic algorithm performed similarly to the template matching comparison with an average RMS error of approximately 3.17 pixels, and a continual trend of improvement over the course of the 15 minute period. The rough-feedback actor critic algorithm hovered around the same average RMS error of 3.17 pixels and followed the general trend of improvement exhibited by its classical counterpart. Both the Q-learning and sarsa algorithms performed better using the average grey level tracking method as seen comparing their average RMS errors from Fig. 6.5 and Fig. 6.6. This was not expected as the error in positioning with the average grey level method was expected to have been greater than the template matching technique. One possibility related to the plots is that the initial states were significantly better than those of the template matching experiment. This can be seen by examining the first samples and noting how much less error is present at that point already. The improved starting conditions could have been related to experimental setup as coordinating the experiments was setup and easily affected by human error. The rough-feedback Q-learning method behaved as expected, exhibit-

ing slightly more average RMS error than the original template matching experiment did. Similar to the previous experiments, rough-feedback sarsa maintained a comparable performance with the classical algorithm, maintaining a slightly higher error of approximately 3.0 pixels compared to 2.8. The results for the rough-feedback methods each contained approximately 300 less samples due to the extra computational requirements needed for generating the rough coverage values and adjusting the learning rates.

For most cases presented in this section, the template matching tracking method provided the same or better accuracy with nearly the same amount of samples or speed as the average grey level tracking technique. As a result, template matching appears to be the most favourable approach for target tracking at this point. Examining the results for each of the tracking algorithms indicated that the trade-off associated with extra computations required by the RL methods paid off in accuracy. Rough-feedback Q-learning was the most consistent performer suited to the task of target tracking in each of the different lengths of time and tracking methods. This was not a surprising result since Q-learning exhibits a more deterministic approach in selecting future actions in any given state due to the nature of the algorithm. Since the environment was relatively static with minimal external noise, a

more greedy approach should tend to do better. To provide a more rigorous view of algorithm performance, several additional experiments were included to place more demands on the tracking system to uncover the response.

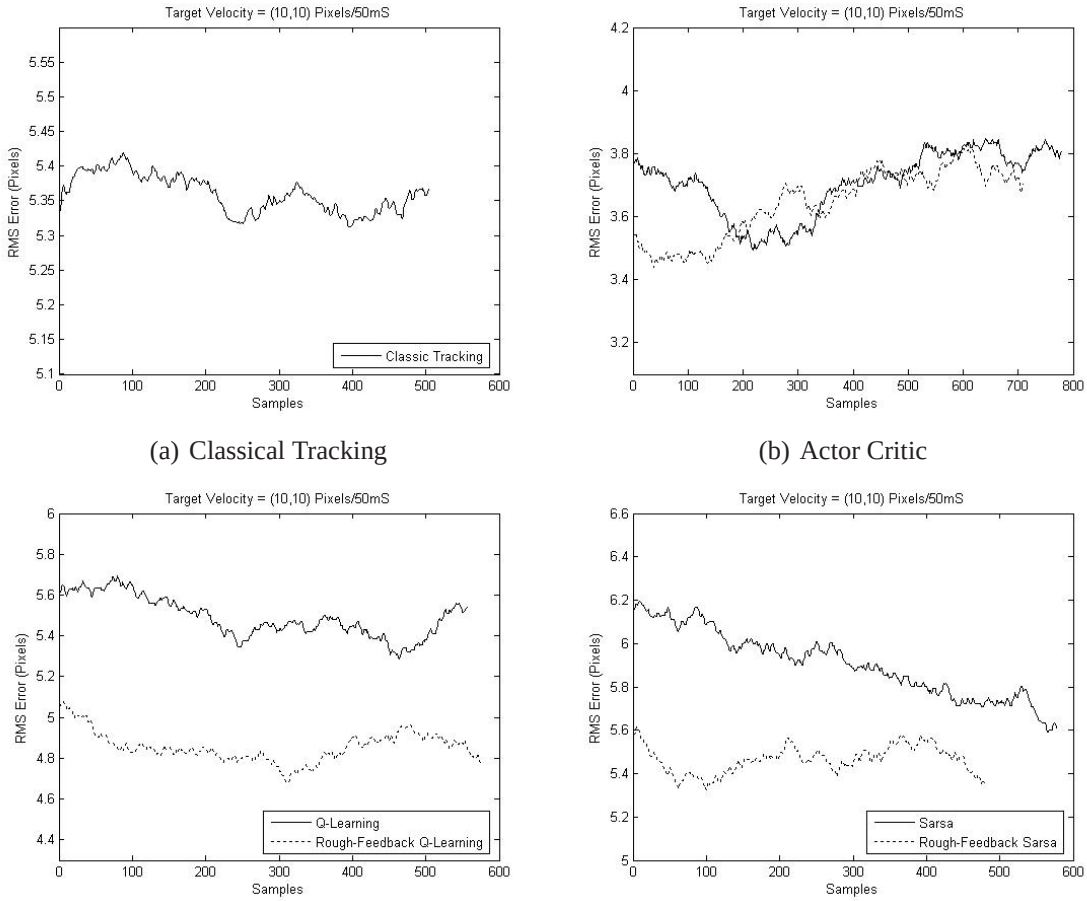
6.1.2 Variable Target Speed

The original time varied experiments maintained a speed of (6,6) pixels of movement per 50mS. This represented the movement in both the x and y-directions of the target. This rate of movement was sufficient to make the tracking tests meaningful but not fast enough to out-hustle the tracking algorithms. Since actual conditions that the line crawling robot may face could include high winds commonly found on the prairies, faster moving targets are a real possibility. In order to provide some insight into performance in these types of conditions, the speed of the target was increased in successive steps. Results are included for a speed of (10,10) pixels of movement per 50mS for comparison with the results from the previous section where movement was restricted to (6,6) pixels/50mS. This consisted of an increase of movement rate from 120 pixels/second up to 200 pixels/second. The extra speed may not be enough to force the algorithms to drop the target, but

it was expected to expose performance related issues for situations when faster tracking is required.

Two sets of results are included for this experiment, one for the template matching technique, and one for the average grey level tracking method. Besides the tracking methods and the various algorithms, the remaining parameters for the test vectors remained static, including the distance from the target (17cm), height of the target (4.25cm), target velocity (10,10) pixels/50mS, length of each experiment (set at 5 minutes) and the target movement pattern (used a clockwise circular movement pattern). Additional tests beyond (10,10) pixels/50mS were investigated, but the distance travelled by the target in 1 second was enough to move completely out of the field of view of the camera causing some of the slower tracking methods to fail completely by losing sight of the target. The results presented in this section contain the first speed increment (from (6,6) up to (10,10)). Five minute lengths were selected as they provided enough time for learning methods to settle and begin learning how to track the target if possible.

Template matching tracking was the first method tested including all seven algorithms. The results are presented in Fig. 6.7. As expected, each of the tracking algorithms exhibited an increased average RMS error. Inter-



(a) Classical Tracking

(b) Actor Critic

(c) Q-Learning

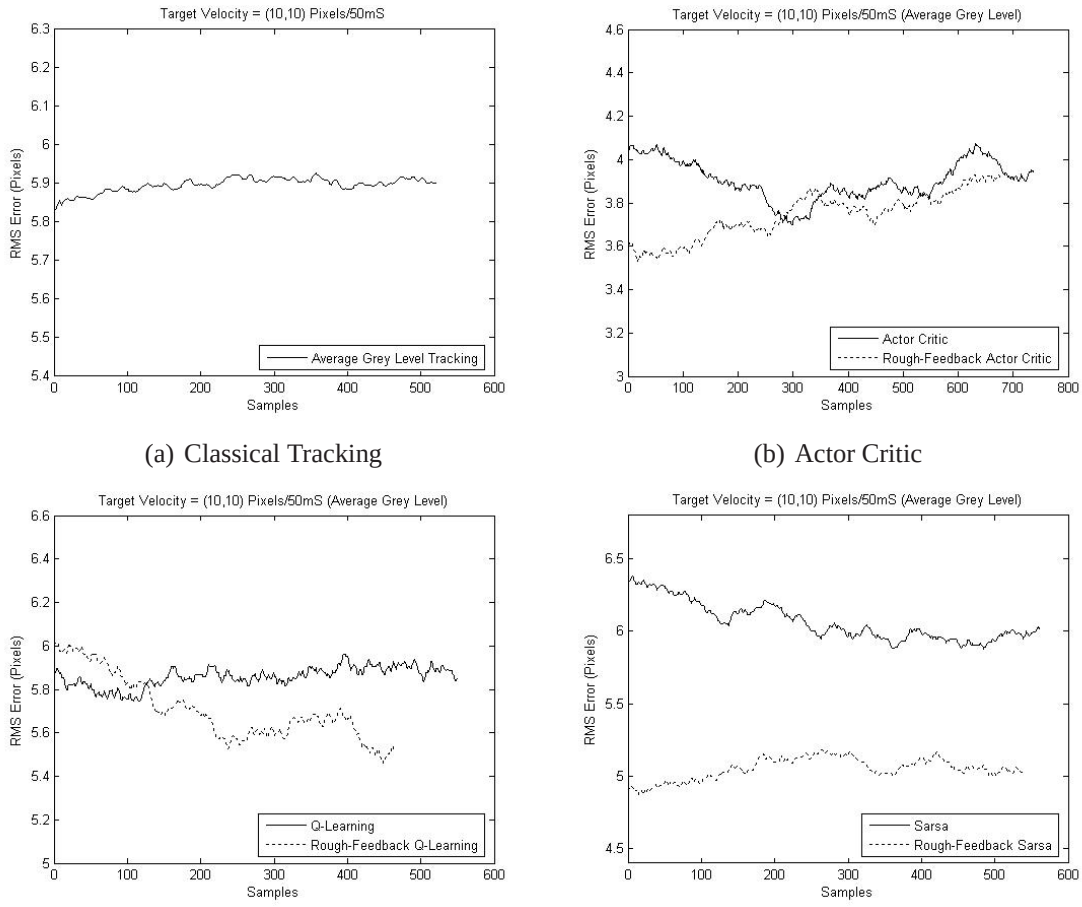
(d) Sarsa

Figure 6.7: Template matching, 5-minute high speed (10,10) tracking experiment average RMS error results

estingly, the actor critic algorithm outperformed the remaining algorithms, displaying the smallest increase in error up to approximately 3.7 pixels. The classical Q-learning algorithm was the only learning method that had worse performance than classical target tracking. The remaining learning methods either decreased or maintained an average RMS error similar to the initial value over time, as seen for Q-learning and sarsa. Both Q-learning and sarsa had significant drops in target accuracy at the increased target speed. The most likely reason for the drop in performance is based on how the algorithms learn action values. The entire set of possible actions are scanned to determine which is most suitable in any given situation, requiring time to do so. At the point when the desired action was detected, the target had since moved and the error rate increased. A method like the actor critic approach with a separate policy can help avoid the need for scanning the entire action space as it will dictate which actions to take using the policy, so once it has become experienced, the potential for reducing the amount of hunting for appropriate actions is very likely.

The average grey level tracking method was used to generate a second set of results. The parameter values were maintained the same to provide a meaningful comparison of the two methods in addition to either provid-

ing further evidence to support or refute the observations from the template matching technique. The results can be seen in Fig. 6.8. The classical track-



(a) Classical Tracking

(b) Actor Critic

(c) Q-Learning

(d) Sarsa

Figure 6.8: Average grey level tracking, 5-minute high speed (10,10) tracking experiment average RMS error results

ing algorithm demonstrated an increased average RMS error of 5.9 pixels compared to 5.36 using template matching, confirming the presence of in-

creased error expected with average grey level tracking. Once again, the actor critic method was least affected by the increased target velocity, confirming the expected results. The average RMS error values for all of the learning methods were slightly increased, accounted for by the switch to the more error-prone tracking method. The only exception was rough-feedback sarsa which exhibited improved performance (by 0.3 pixels) likely attributed to favourable initial conditions. In each of the RL methods, the rough-feedback performance was equal to or better than the classical algorithm interpretation. This was surprising as the extra processing time required to generate the rough coverage values was expected to incur some extra position error due to a faster moving target. The adjustable learning rate appeared to more than compensate for the slower rough coverage algorithms by revising the policy correction action appropriately to help find the best policy. This flexibility would allow for a quicker and more reliable means of maximizing the action values of any given state.

Speeding up the target velocity demonstrated how each algorithm and tracking technique would respond. Not surprisingly, the algorithm with the least amount of required processing time was the best performer. From a practical perspective, the actor critic algorithm would be most suited to

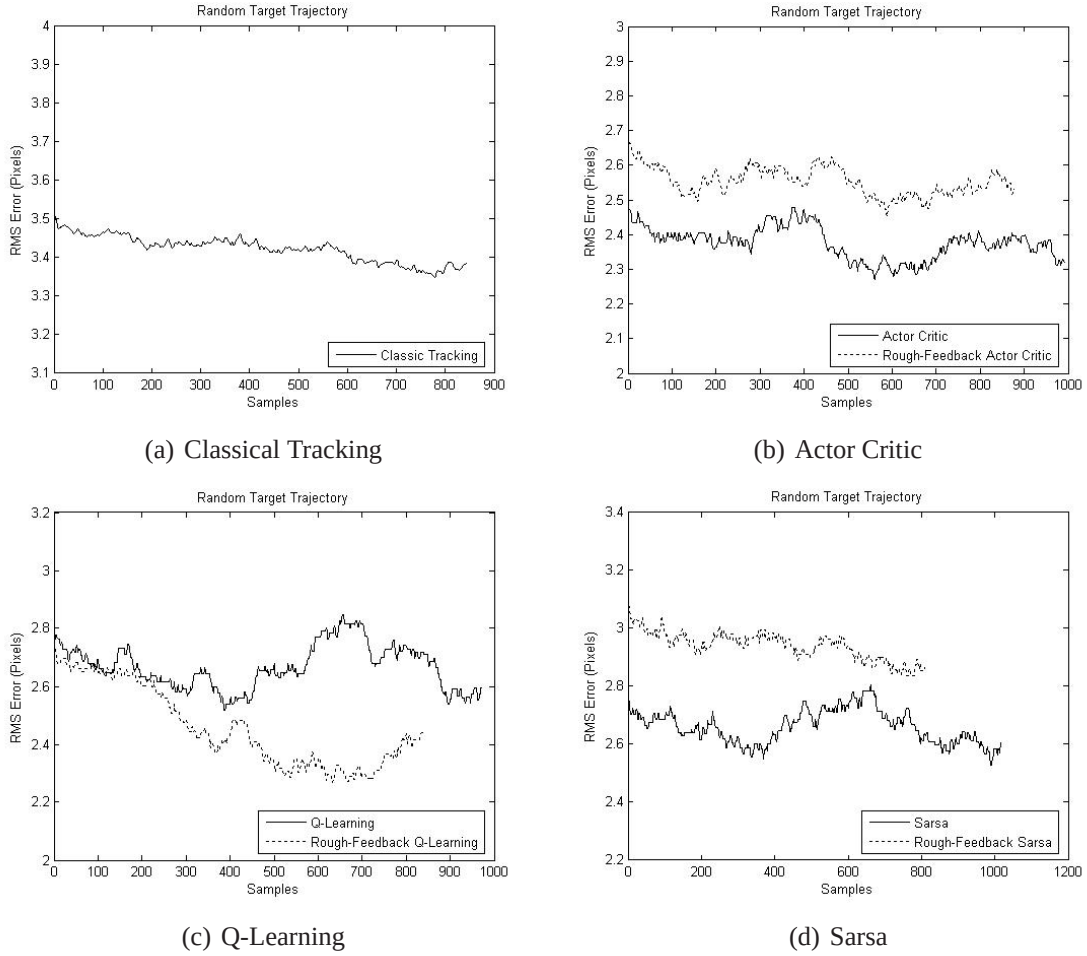
faster moving platforms or high wind conditions where quick actions were necessary to acquire the best images. Once again, template matching provided suitable speed and improved accuracy over the average grey level tracking method, making it more favourable in its current configuration.

6.1.3 Random Target Trajectories

The experiments up to this point were done using a clockwise, circular trajectory for the target pattern. In the event that the RL methods might have been gaining some advantage in learning the shape and its movement characteristics, random movements were employed to see how the algorithms fared when there was no standard movement pattern. This would also help provide some insight into how short sharp movements with the line crawler, such as during travel, operations on the line or even in gusting wind conditions or variable illumination might be dealt with.

There were two variable parameters for this experiment, the tracking method, and the associated algorithm. The rest were fixed, including the distance to the target (17cm), height of the target (4.25cm), target velocity (6,6) pixels/50mS, target trajectory pattern (random), and the time duration for each experiment (5 minutes). The template matching results are

shown in Fig. 6.9. The results in Fig. 6.9 showed that the learning methods



(a) Classical Tracking

(b) Actor Critic

(c) Q-Learning

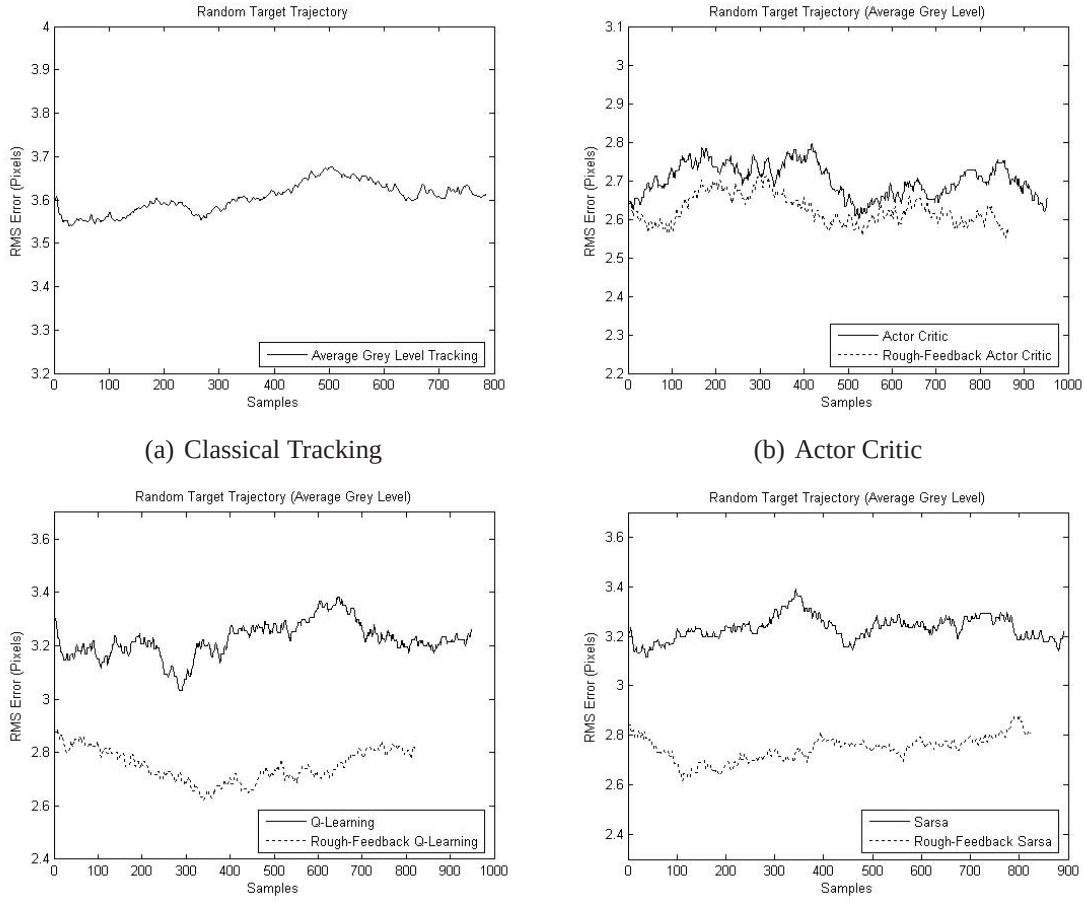
(d) Sarsa

Figure 6.9: Template matching tracking, 5-minute random trajectory tracking experiment average RMS error results

were not fooled by random moving targets compared to a standard pattern. In fact, the average RMS error of each method improved over the circular clockwise trajectory. The reason why all of the tracking methods are more

well suited for random movements is due to the fact that once the outer edge of the available range was met, the target tended to return toward the centre for at least a short duration before moving in a different direction and at this point, the camera did not need to move much to centre the target. The circular trajectory on the other hand continued moving in a wide enough pattern that continual tracking was needed to centre the target in the field of view. Even the classic tracking algorithm without learning exhibited an improved performance over the original 5 minute test seen in Fig. 6.3 the original error was 4.2 pixels compared to 3.45 shown in Fig. 6.9.

The average grey level tracking method was included in this test to provide additional evidence of the performance using targets with random trajectories. The results were expected to provide similar improvements as template matching but with slightly more error based on the nature of the technique. After examining Fig. 6.10, the results for each algorithm showed slightly worse accuracy as expected due to the nature of the tracking method. The classical tracking algorithm provided the best base comparison as there was no potential advantage due to learning. The average RMS error for classical tracking using template matching was approximately 3.45 pixels, compared to 3.6 pixels for the average grey level approach (an consistent



(a) Classical Tracking

(b) Actor Critic

(c) Q-Learning

(d) Sarsa

Figure 6.10: Average grey level tracking, 5-minute random trajectory tracking experiment average RMS error results

error increase of about 4%).

The target tracking system responded more favourably to targets with random movement trajectories. The original circular trajectory used for all other tests continually moved along the widest circumference possible, covering the most ground and making it more difficult to track. The random trajectory target movements were often interior to the maximum circumference, making it less difficult to centre in the field of view and simulate more closely actual target movement as circular motion would be fairly uncommon in field trials. The most accurate algorithms were rough-feedback Q-learning and the classical actor critic implementations. Both of these techniques coupled together with template matching tracking provided an average RMS error rate of approximately 2.4 pixels. The difference between them was the reduced amount of samples provided by the rough-feedback Q-learning method, containing 140 less samples than the classical actor critic implementation.

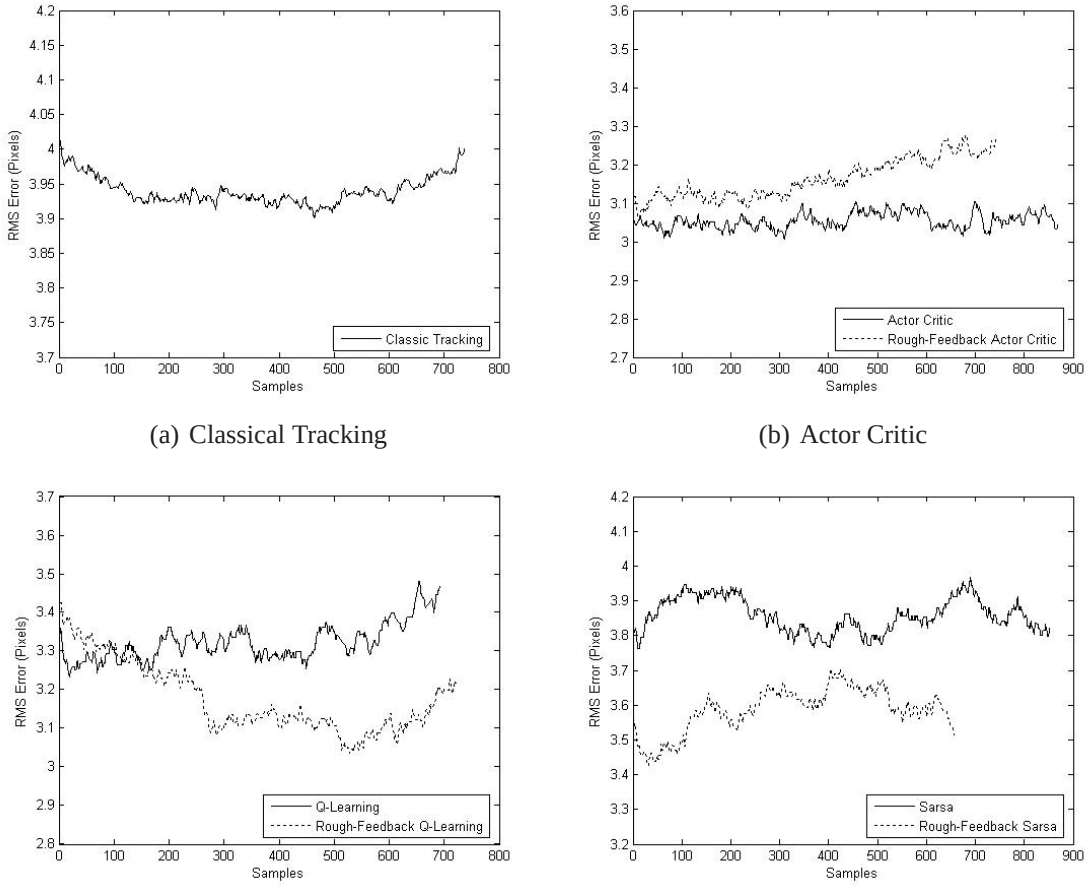
6.1.4 Noise Susceptibility

Since there are a number of possible sources of noise that could affect the line crawling robot, such as atmospheric, man-made, galactic, or even in-

ternal systems, the goal of this experiment was to examine the performance for each of the algorithms and tracking methods when subjected to input noise. As mentioned briefly (see Sec. 5.4), a uniform deviate was used to add random noise to the target location before presenting the information to the tracking algorithm. Besides the tracking techniques and algorithms, all remaining parameters for the test vectors were kept constant during this experiment, including distance to the target (17cm), height of the target (4.25cm), target velocity ((6,6) pixels/50mS), target trajectory (clockwise circular pattern), the length of time (5 minutes), the chance of noise present at any given iteration (50%), and the possible range of noise magnitude in camera movement steps (+/- 5).

The results expected for noisy environments favoured the learning methods, as they were able to make adjustments to the actions in any given state over time they should offer improved performance over the classical tracking techniques learning to become more conservative in movements. Out of the collection of learning algorithms, the actor critic has been most resilient up to now and its efficiency in dealing with rapid change made it the favourite for best results prior to running the experiments. The results are presented in two parts again, first the template matching technique

(see Fig. 6.11) followed by the average grey level tracking method (see Fig. 6.12). After examining the results in Fig. 6.11, the performance predic-



(a) Classical Tracking

(b) Actor Critic

(c) Q-Learning

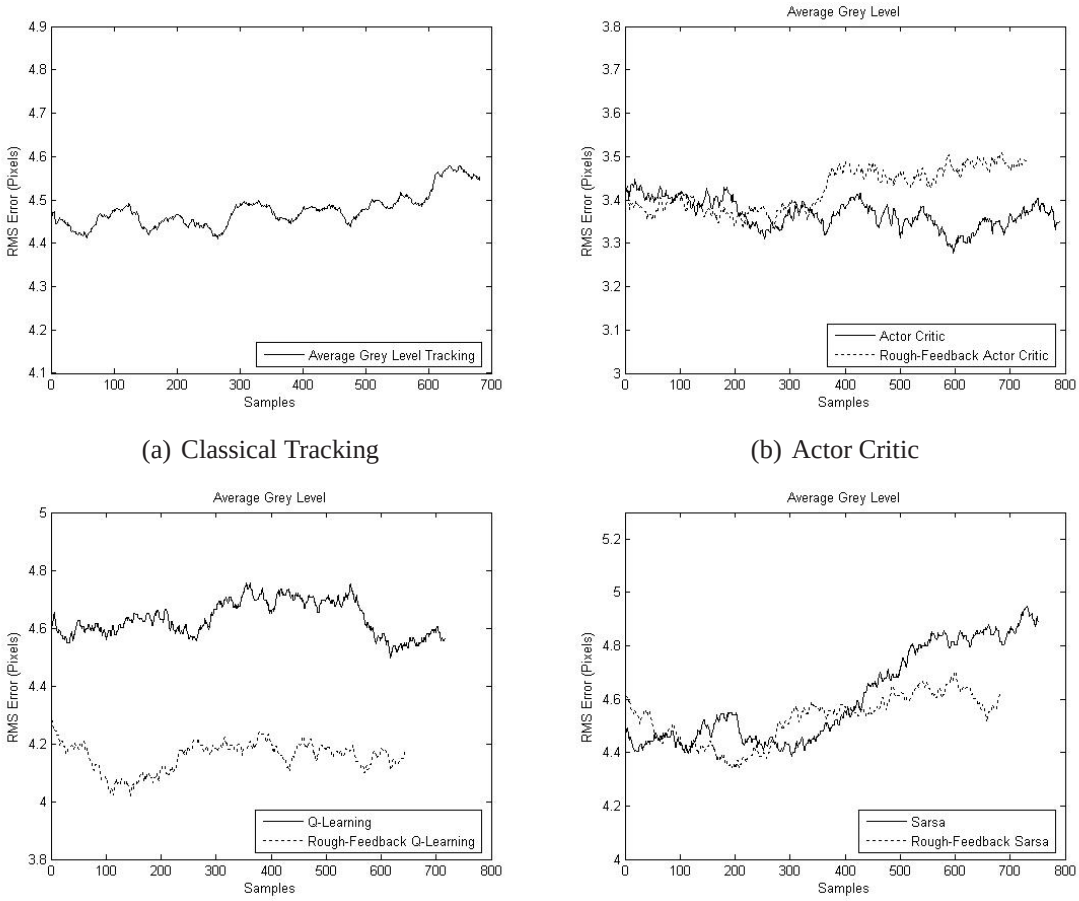
(d) Sarsa

Figure 6.11: Template matching tracking, 5-minute noise susceptibility tracking experiment average RMS error results

tions were not far off. The tracking algorithm least affected by noise was the actor critic method, exhibiting an average RMS error of approximately

3.05 pixels. Surprisingly, rough-feedback Q-learning performed comparably well in the noisy environment, yielding an average RMS error of 3.18 pixels. The greedy nature of both Q-learning algorithms saw them outperform their sarsa equivalents. As expected, the classical algorithm performed the worst, with an average RMS error of 3.95 pixels as it was unable to compensate for input noise.

The average grey level tracking method was included in this test to gather additional evidence of performance in noisy environments. The results were expected to register more error than the template matching technique due to the associated higher rate of error compounded together with the newly injected noise. From Fig. 6.12, the initial expectations can be confirmed, the error associated with the average grey level tracking method together with the additional noise increased the average RMS error of each tracking algorithm. However, once again the actor critic method provided the best performance, with an average RMS error of 3.35 pixels. The RL algorithms based on learning action values showed a significant increase in error, approaching and in some cases exceeding that of the classical tracking algorithm. Q-learning, sarsa, and rough-feedback sarsa all exhibited worse performance than the classical tracking algorithm that scored 4.48 pixels of



(a) Classical Tracking

(b) Actor Critic

(c) Q-Learning

(d) Sarsa

Figure 6.12: Average grey level tracking, 5-minute noise susceptibility tracking experiment average RMS error results

average RMS error. Rough-feedback Q-learning was the only action value learning method that performed better, averaging just under 4.2 pixels of error. The results show that the extra associated error with the average grey level tracking method make it less likely to perform well in a noisy environment.

Target Tracking Results

Template Matching - Average RMS Error (Pixels)

Algorithm	1-min Test	5-min Test	15-min Test	Random Trajectory	High Speed Target	Additive Noise
Classical Tracking	3.8138	4.1905	3.829	3.4228	5.3605	3.9404
Actor Critic	3.5107	3.1835	3.0605	2.3757	3.7063	3.0543
Q-learning	4.2086	3.3548	3.4794	2.6673	5.4906	3.3225
Sarsa	3.276	3.6599	3.5874	2.6604	5.9056	3.8578
Rough-Feedback Actor Critic	3.3628	3.1301	3.0509	2.5526	3.6433	3.163
Rough Feeback Q-learning	3.9593	3.1384	2.8454	2.4576	4.8577	3.1773
Rough Feedback Sarsa	4.4166	3.7095	3.5751	2.9405	5.4647	3.5866

Average Grey Level - Average RMS Error (Pixels)

Algorithm	1-min Test	5-min Test	15-min Test	Random Trajectory	High Speed Target	Additive Noise
Classical Tracking	4.0055	4.3541	3.7742	3.7748	5.8966	4.4762
Actor Critic	3.3803	3.1646	3.0646	2.7017	3.9041	3.3685
Q-learning	3.6675	3.5438	3.2154	3.2229	5.8582	4.6324
Sarsa	3.6044	3.4415	2.807	3.233	6.0655	4.61
Rough-Feedback Actor Critic	3.2405	3.1756	3.0541	2.6245	3.7421	3.4247
Rough Feeback Q-learning	3.903	3.2413	3.1237	2.7506	5.7086	4.155
Rough Feedback Sarsa	2.878	3.3866	3.0223	2.7463	5.0526	4.5269

Figure 6.13: Compiled target tracking results

The target tracking system, specifically using template matching and the actor critic algorithm, responded well to a noisy environment. The maximum possible movement allowed in any state was eleven steps, providing noise with a magnitude of five steps had the potential to easily confuse the tracking methods, causing more average RMS error. This can be seen read-

ily in the classical algorithm results as well as the action value learning methods. Average grey level tracking compounded its own increased error with that of the input noise and exhibited greater susceptibility to high error rates making it a less favourable choice for a noisy environment. Besides selecting template matching, the two algorithms most suited for target tracking in a noisy environment include the classical implementation of the actor critic algorithm for its speed and accuracy and the rough-feedback Q-learning algorithm for its accuracy. The compiled results can be found in Fig. 6.13.

6.2 Line Crawling Experiments

The last stage of experimental work was to test the functionality of the line crawling robot. At the time of writing this report, the line crawlers operated as individual entities when crawling on the skywire. As a result, the functional tests included simple locomotion, climbing up inclines, and acquiring images of targets. The tests and results were all performed indoors using the manufactured experimental environment described in Sec. 5.1.

The first functional testing experiment was simple locomotion along the skywire. The support and drive motors were the components under load

during these tests, demanding the support of a 1.6kg robot and locomotion along a section of skywire. The experimental setup can be seen in Fig. 6.14, showing a rear-view of the line crawling robot in its natural habitat. The line crawling robot was able to navigate back and forth along the

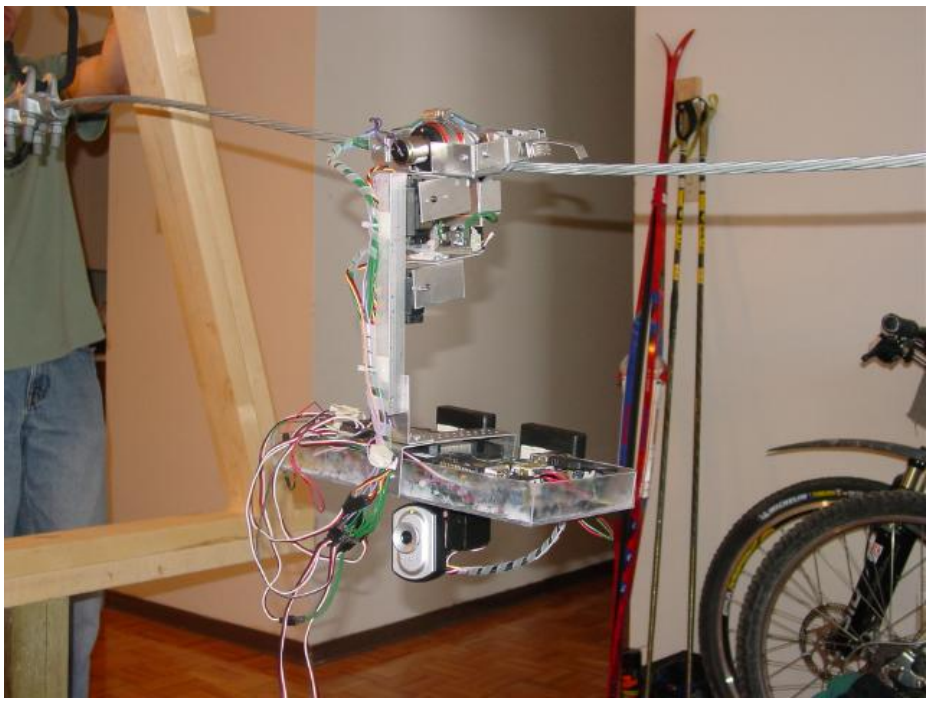


Figure 6.14: Simple locomotion experiment in progress

skywire supporting the entire weight of the robot without any difficulties. The speed of travel was quite slow due to the high gear ratio in use, resulting in 0.00165m/s, covering approximately 10cm of distance in one minute. Using such a slow motor with strong torque provided a very sturdy drive

useful for navigating up and down varying inclines of skywire.

The next functional testing experiment involved varying the angle of inclination of the skywire. During the design phase, slopes of 45 degrees were specified and thus expected to be suitable. Fifteen and thirty degree inclines were attempted first with good success. The extra traction provided from the rubber-lined wheels helped the line crawler navigate up the slopes without much difficulty (see Fig. 6.15). The section of line used was 3.6 metres in length and the slope was set by adjusting the proximity of the support towers. The angle of inclination was measured at the steepest point near the tower structures. The 30 degree incline test was successful but introduced a problem based on the rigidity of the line grip. The weight is pulled straight down by gravity and with a rigid line grip and a weight distribution with two-thirds of total in the platform, the drive wheel furthest from the direction of travel up the incline slowly lost contact, this can be seen in Fig. 3.16. Even though the trailing drive wheel was losing contact, the robot was still able to power up the incline with no difficulties. The 45 degree incline exhibited a more exaggerated problem of losing contact, but the traction and torque were still significant enough to climb the incline. To address this problem, some degree of movement for the payload would allow the line

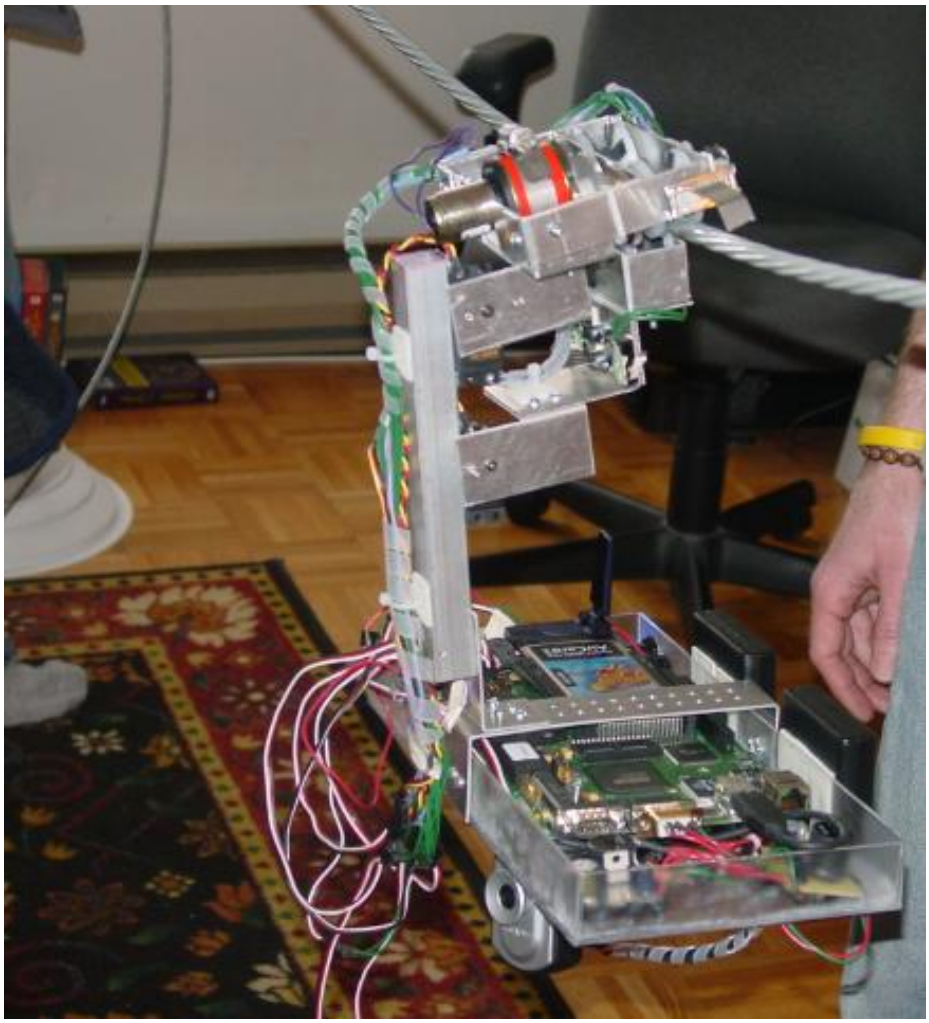


Figure 6.15: About to climb a 15 degree inclined section of skywire

grip to form to the incline of slopes while the platform remained parallel to the gravitational force.

The last part of functional testing included acquiring images during travel on the skywire. Through additional work being done on the TS-5500 in the CILab group, wireless communication was established through the use of an air card in the PCMCIA slot, which can be seen in Fig. 6.16 which shows the line crawler positioning to acquire an image of a sample insulator on the floor. With a communication link established to the line crawling robot, it was possible to issue commands to establish the behaviours discussed in Sec. 3.12. The base behaviour was obstacle avoidance and locomotion along the skywire. The higher level behaviour pertained to image acquisition of desired targets. For the purposes of this experiment, the target consisted of the sample insulator seen in Fig. 6.16. Once the target had been spotted, the line crawling robot moved into position and acquired the image (see Fig. 6.17). At the time of writing, only the first layer of behaviour was automated. Acquiring images was achieved through a sensor interface. This was accomplished through triggering the micro-switches on either side of the line crawler. The thinking behind this was that when an obstacle is encountered, usually it is near a tower as that is where the vibration dampers and

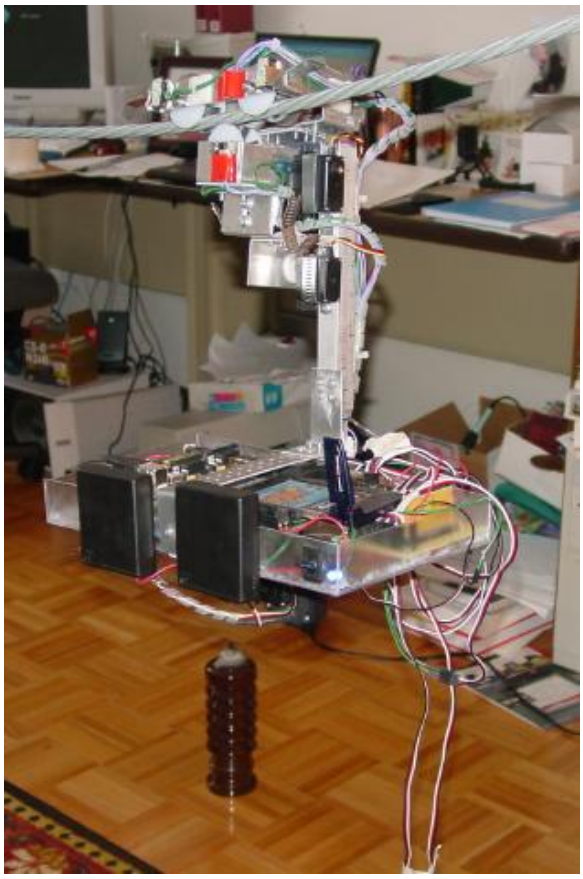


Figure 6.16: The line crawler positioning to take a picture of a sample insulator

Figure 6.17: Line crawler view of the target insulator

line clamps are located (see Fig. 2.1). The towers coincide with the location of the insulator stacks and the insulators were the primary targets of interest. When one of the micro-switches was triggered, the line crawler would move in the opposite direction for 5 seconds before halting and scanning for targets. The location of the sample insulator was placed in a convenient location in a well lit environment so that the view of the camera would only see one dark object, allowing the average grey level tracking technique to be implemented without worrying about templates for tracking. Once a target was located in the goal state (S_4), an image can be acquired such as the one seen in Fig. 6.17. Since both the line crawler and the target were essentially stationary, acquiring the image was a relatively stable process.

6.3 Summary

After generating all of the results from the various test vectors and functional testing, a strong picture of the performance of the line crawling robot was formed. Each experiment targeted different aspects of the line crawling robot with the intent of proving the utility of the sub-system as part of the entirety. The process was to exercise as many aspects through refining test vectors and selecting the appropriate values to provide concise experiments with meaningful results without having redundancy or too much overlap. The ideas and information discovered during the experimental stages were drawn upon to form the the conclusions found in the next chapter.

7 Conclusions and Recommendations

In this report, the design and development of a line crawling robot were outlined. This included both the hardware and software perspective of the line crawler based on its goal of travel along skywire and acquiring useful images of selected targets. Through the process of design, verification and experimental work, the operation and performance were confirmed and measured. The three main aspects of this thesis included the design and development of the line crawling robot, the target tracking problem using a monocular vision system and functional performance on skywire leading to autonomous inspection.

7.1 Conclusions

In its current capacity, the line crawling robot is capable of navigating safely along sections of skywire in search of desired targets. Besides being able to support its own weight and move around, it has the capacity to interface with a number of different devices through USB or PCMCIA ports, allowing for more advanced capabilities including communication and being able to send live or previously stored images. Although the physical design is currently

meant for individual units, preliminary support was added to the design for future expansion into a cooperative platform capable of performing more advanced tasks such as circumnavigating obstacles and sharing information with one another. The adjustable nature of the line grip allowed an opening and closing action where the open position avoids contact with obstacles, allowing the line crawler to avoid objects in its path with the support of other line crawling robots, similar in nature to the first Lego®prototype robot. All of the motors were specified to support double the weight of a single line crawler with the future intent of docking with and helping support additional robots for a limited amount of time to achieve more complicated tasks.

The target tracking problem was specified and a solution was developed with a monocular vision system. Two degrees of freedom allowed for panning and tilting the camera to centre desired targets in their field of view before acquiring important images. Two tracking methods were examined along with seven algorithms. The template matching technique provided improved accuracy in almost all cases with all algorithms. The operating cost was expected to be higher, reducing the number of samples over the experimental period however the results proved negligible making it a

more appealing choice over the average grey level tracking method. The one saving grace for the average grey level method during the experimental work was that it did not require any previous knowledge of its target except that it must be darker than its surroundings. The size of the target, the orientation and shape were not as important. The algorithms implemented demonstrated that the RL techniques provided a suitable choice for target tracking in all conditions. Specifically, the classical implementation of the actor critic method and the rough-feedback Q-learning methods were the top performers for reduced error rates when tracking a target. This was tested in conditions with varied time, speed, target trajectory and susceptibility to random noise. The actor critic method excelled due to the quick nature of its ability to select actions as it stores its own policy instead of learning action values. On the other hand, the rough-feedback Q-learning method excelled due to its greedy nature and the adaptable learning rate based on the rough coverage as a performance metric, altering the step size in the update phase of the learning process based on previous performance.

Functional operation of the line crawling robot consisted of simple locomotion and support of its own weight, navigating up and down inclines and acquiring images of desired targets. Through verification and experi-

menting, basic functionality of the line crawling robot was refined to the point of being attached anywhere along the skywire and navigating a complete section. During navigation, behaviour of the robot followed a simple hierarchical plan with survival and obstacle avoidance being the first strategy and target image acquisition being the second strategy. To improve the chances of acquiring meaningful images, the line crawling robot stopped to take pictures, reducing the error rate in positioning the centre of focus for the camera on the centre of the target. For the preliminary tests, targets of interest were placed alongside the ends of the section of skywire near the line clamp obstacles. This combined the behaviour of obstacle avoidance and image acquisition. Although the current design lacked flexibility in adjusting to the form of sloped lines, the power and torque necessary to climb a 45 degree inclined section of line was available with the addition of rubber lined wheels and a slow speed motor with high torque.

7.2 Recommendations

Each of the areas of the line crawling robot discussed in the conclusions contained room for expansion due to the broad scope of the project. The topics are included along with suggestions for future work and recommen-

dations.

To extend the functionality of the line crawling robot and achieve a wider range of goals during operation on the skywire there are several areas that could be pursued. Returning to the structural design of the robot, providing some degree of flexibility in motion of the payload would allow the line grip to form more closely to the shape of the skywire on steeper inclined sections. Another important step for future work is the addition of a means to recharge the power supplies. The batteries selected were rechargeable by nature but the means to restock them with power was not available without human assistance. Some efforts were already in place to discover if a solar trickle charger might solve the problem. The idea of flexible, flat-panel solar cells were entertained but more testing needs to be done to find out how NiMH batteries respond to trickle charging or if switching to a different variety would be more well suited. Another area for expansion would be providing more sensory equipment for proximity detection of obstacles. The preliminary testing and format of the line crawling robot did not include this type of device, however moving to a multiple robot platform would benefit from being able to detect the proximity of an obstacle, either fixed or mobile without having to physically contact it, preventing potential mishaps

or inter-robot damage. This can be achieved with sonar or IR proximity detectors.

There were a few areas of work underway on the tracking system at the time of writing. The tracking methods were suitable for operating in controlled environments, however extension to targets of varying size, shape, orientations and in various locations would cause both methods to perform worse. Efforts were being put into discovering improved descriptors that are more robust for locating targets of interest without needing exact templates or stark background contrasts to locate them. Both the average grey level tracking method and the action-value learning RL methods were being examined with the intent of streamlining the processes. The poor performance of average grey level tracking warranted a review of the implementation to ensure there were no redundancies in the code and some simple optimization ideas were discovered, such as removing the need for the division operation as the total of each cell would still be a multiple of the total number of pixels, accurately depicting which contained the lowest grey level. The action value learning methods, specifically the rough-feedback versions were interesting as they performed well in trials and the goal was to improve the speed of the tracking method. The improvements currently under investigation included

monitoring the action values at each state and only revising the learning rate when worse results were being registered. This would have an expensive initial startup cost as the starting performance is usually worse than after a few revisions take place, but the later improvement could present a potential benefit.

Basic functionality was achieved providing security on the skywire for the line crawling robot, however there were several areas of improvement for future work. Extension to multiple line crawlers would allow a number of extra tasks and autonomy for a group of robots that a single unit could not achieve. Docking facilities for one robot to connect to another could potentially allow navigation around awkward obstacles and potentially move robots to safety when remaining power needs to be conserved. For safety of the people involved, the experimental work took place on the reduced size tower structure, without live conductors. The next step in proving the design would be to move to field trials above live power lines. Testing to discover the response of the line crawling robot and all of its onboard systems in close proximity of the high noise environment provided by live conductors may warrant the use of shielding. Although the line crawler was built with the intent of operating over three seasons, (excluding winter) weather proofing

the design for outdoor operation is another area that needs to be addressed. Ideas such as a rain shield and coating all exposed circuit boards have been entertained. Further investigation into lightning storms and being able to predict what would happen to the line crawler and steps to take in avoiding danger are necessary since the skywire is intended to provide a path to ground for lightning strikes.

Index

References

[1] J. E. A. Bertram, and Y. Chang, 'Mechanical Energy Oscillations of Two Brachiation Gaits: Measurement and Simulation', *American Journal of Physical Anthropology*, vol. 115, pp. 819-826, 2001.

[2] M. J. Beynon, B. Curry, and P. Morgan, 'Knowledge discovery in marketing: An approach through Rough Set Theory', *European Journal of Marketing*, vol. 35, no. 7-8, pp. 915-937, 2001.

[3] M. Borkowski and C. Henry, *CILab: personal communication*, 2006.

[4] R. A. Brooks, 'A Robust Layered Control System for a Mobile Robot', *IEEE Journal of Robotics and Automation*, RA-2, pp. 14-23, April 1986.

[5] R. A. Brooks, 'Planning is Just a Way of Avoiding Figuring Out What to do Next', *Technical Report, MIT Artificial Intelligence Laboratory*, Working Paper 303, Chapter 6, pp. 103-110, September 1987.

[6] R. A. Brooks and A. M. Flynn, 'Fast, Cheap and out of Control: A Robot Invasion of the Solar System', *Journal of The British Interplanetary Society*, Vol. 42, pp. 478-485, 1989.

[7] J. Brown, *Brief H-Bridge Theory of Operation*, Dallas Personal Robotics Group, 1998, http://www.dprg.org/tutorials/1998-04a/ *Last checked for content, February 1, 2007.*

[8] I. Buchman, *The nickel-based battery, its dominance and the future*, Battery University, 2005, http://www.batteryuniversity.com/partone-4.htm *Last checked for content, February 1, 2007.*

[9] D. Clark, and M. Owings, *Building Robot Drive Trains*, New York, NY: McGraw-Hill Inc, 2003.

[10] J. H. Connell, 'A Behaviour-Based Arm Controller', *MIT Artificial Intelligence Lab*, AI Memo 1025, pp. 1-15, June 1988.

[11] V. Degtyaryov, *Design of a Line Crawling Robot with an Intelligent Hybrid Control System*, M.Sc. Thesis, Department of Electrical and Computer Engineering, University of Manitoba, 2003.

[12] *Elan SC520 Data Sheet*, Advanced Micro Devices, 2001, http://www.amd.com/ *Last checked for content February 1, 2007.*

[13] *QRD1113/1114 Reflective Object Sensor Datasheet*, Fairchild Semiconductor, 2006, http://www.fairchildsemi.com/ds/QR/QRD1114.pdf *Last checked for content, February 1, 2007.*

[14] C. Gaskett, *Q-Learning for Robot Control*, Ph.D. Thesis, Department of Systems Engineering, The Australian National University, 2002.

[15] R. C. Gonzalez and R. E. Woods, *Digital Image Processing: Second Edition*, Prentice Hall Inc., Upper Saddle River, New Jersey, 2002.

[16] I. M. Gotlieb, *Regulated Power Supplies: Fourth Edition*, McGraw-Hill, Blue Ridge Summit, PA, pp. 327-444, 1992.

[17] J. Hawkins with S. Blakeslee, *On Intelligence*, New York, NY: Times Books, Henry Holt and Company, 2004.

[18] C. Henry, *Reinforcement Learning in Biologically-Inspired Collective Robotics: A Rough Set Approach*, M.Sc. Thesis, ECE Department, University of Manitoba, January 2006.

[19] J. Herbert and J. Tao, 'Time-Series Data Analysis with Rough Sets', *4th International Conference on Computational Intelligence in Economics and Finance (CIEF)*, Salt Lake City, USA, pp. 908-911, July 2005.

[20] B. K. P. Horn, 'Exact Reproduction of Colored Images', *Computer Vision, Graphics and Image Processing*, vol. 26, pp. 135-167, 1984.

[21] T. Jaakkola, M. I. Jordan, and S. P. Singh, 'Convergence of Stochastic Iterative Dynamic Programming Algorithms', *Neural Computation*, 6, pp. 1185-1201, 1994.

[22] B. Jahne, *Digital Image Processing: 6th revised and extended edition*, Springer-Verlag, Berlin, Germany, pp. 31-41, 2005.

[23] L. P. Kaelbling, M. L. Littman, and A. W. Moore, 'Reinforcement Learning: A Survey', *Journal of Artificial Intelligence Research*, vol. 4, pp. 237-248, May 1996.

[24] B. Knudsen, *CC5X C Compiler*, B. Knudsen Data, 2006, http://www.bknd.com/ *Last checked for content February 1, 2007.*

[25] J. Komorowski, L. Polkowski, and A. Skowron, 'Rough Sets: A Tutorial', In S.K. Pal and A. Skowron, editors, Rough-Fuzzy Hybridization: A New Method for Decision Making, Springer-Verlag, Singapore(in Print), 1998.

[26] D. Lancaster, 'The Mount Graham Aerial Tramway', *Midnight Engineering*, May-June 1995, pp. 25-27.

[27] G. Ledwich, *DC-DC Converter Basics*, http://www.powerdesigners.com/InfoWeb/design_center/articles/DC-DC/converter.shtm *Last checked for content, February 1, 2007.*

[28] *Linear Technology: LTC1383 Datasheets*, 1994, http://www.linear.com/ *Last checked for content, February 1, 2007.*

[29] *Linux RedHat 9.0 Manuals*, 2003, http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/ *Last checked for content, February 1, 2007.*

[30] M. M. Mano, *Digital Design: Second Edition*, Englewood Cliffs, New Jersey: Prentice-Hall Inc, 1991.

[31] F. Martin, *The Handy Board Manuals*, 2000, http://www.handyboard.com/techdocs/index.html *Last checked for content, February 1, 2007.*

[32] G. McComb, *The Robot Builder's Bonanza*, New York, NY: McGraw Hill Inc, 2001.

[33] C. McManis, *H-Bridges: Theory and Practice*, Chuck's Robotics Notebook, 2006, http://www.mcmanis.com/chuck/robotics/tutorial/h-bridge/ *Last checked for content, February 1, 2007.*

[34] *Merkle-Korff Industries*, 2004 http://www.merkle-korff.com/ *Last checked for content, February 1, 2007.*

[35] *Micro Drives: Motor Supplier*, 2005, http://www.micro-drives.com/ *Last checked for content, February 1, 2007.*

[36] *Microchip Technology: Crystal Oscillator Basics and Crystal Selection for rfPIC and PICmicro devices*, Microchip Technology, http://ww1.microchip.com/downloads/en/AppNotes/00826a.pdf *Last checked for content, February 1, 2007.*

[37] *MicroMo Electronics*, 2007, http://www.micromo.com/ *Last checked for content, February 1, 2007.*

[38] *MPLAB Integrated Development Environment*, Microchip Technology Inc., http://www.microchip.com/ *Last checked for content, February 1, 2007.*

[39] T. Munakata and Z. Pawlak, 'Rough Control Application of Rough Set Theory to Control', *Fourth European Congress on Intelligent Techniques and Soft Computing*, vol. 1, pp. 209-218, Aachen, Germany, September 1996.

[40] *National Semiconductor: 3A Step-Down Voltage Regulator Datasheet*, 2007, http://www.national.com/pf/LM/LM2576.html *Last checked for content, February 1, 2007.*

[41] A. Ohrn, 'Discernibility and Rough Sets in Medicine: Tools and Applications', *Ph.D. Thesis*, Department of Computer and Information Science, Norwegian University of Science and Technology Trondheim, 2000.

[42] Z. Pawlak, 'Rough Sets', *International Journal of Computer and Information Sciences* vol. 11, no. 5, pp. 341-356, 1982.

[43] Z. Pawlak, *Rough Sets. Theoretical Reasoning about Data*, Theory and Decision Library, Series D: System Theory, Knowledge Engineering and Problem Solving, vol. 9, Kluwer Academic Pub., Dordrecht, 1991.

[44] Z. Pawlak, 'Why Rough Sets?', *Proceedings of the Fifth IEEE Conference on Fuzzy Systems*, vol. 2, pp. 738-743, September 1996.

[45] Z. Pawlak, 'Rough Set Theory and its applications', *Journal of Telecommunications and Information Technology*, 3, 2002.

[46] J. F. Peters, T. C. Ahn, M. Borkowski, V. Degtyaryov, and S. Ramanna, 'Line-crawling robot navigation: a rough neurocomputing approach', *Autonomous robotic systems: soft computing and hard computing methodologies and applications*, Heidelberg, Germany, Physica-Verlag GmbH, pp. 141-163, 2003.

[47] J. F. Peters, D. A. Lockery, and S. Ramanna, 'Monte Carlo Off-Policy Reinforcement Learning: A Rough Set Approach', *Proceedings of The Fifth International Conference on Hybrid Intelligent Systems*, November 2005.

[48] J. F. Peters, M. Borkowski, C. Henry, D. Lockery, and D. S. Gunderson, 'Line-Crawling Bots That Inspect Electric Power Transmission Line Equipment', *The 3rd International Conference on Autonomous Robots and Agents (ICARA)*, Palmerston North, New Zealand, December 2006.

[49] *PIC16F871 Data Sheets*, 2003, http://ww1.microchip.com/
Last checked for content, February 1, 2007.

[50] PIC Design Catalog, *Technical Section*, pp 13-1 to 13-11, 1997.

[51] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C++: The Art of Scientific Computing: Second*

Edition, Cambridge University Press, University of Cambridge, UK, pp. 278-290, 2002.

[52] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms and Applications*, Prentice Hall Inc., Upper Saddle River, New Jersey, 1996, pp. 784-787.

[53] *Reynolds Electronics: Projects with the PIC Microcontroller*, 1999, http://www.rentron.com/pic.htm *Last checked for content, February 1, 2007.*

[54] B. Rorabaugh, *Mechanical Devices for the Electronics Experimenter*, New York, NY: McGraw-Hill Inc, 1995.

[55] J. K. Rosenblatt, *DAMN: A Distributed Architecture for Mobile Navigation*, Ph.D. Thesis, Robotics Institute, Carnegie Mellon University, 1995.

[56] G. A. Rummery and M. Niranjan, 'On-line Q-learning Using Connectionist Systems', *Technical Report CUED/F-INFENG/TR 166*, Engineering Department, Cambridge University, UK, September 1994.

[57] F. Saito, T. Fukuda, and F. Arai, 'Swing and locomotion control for a two-link brachiation robot.', *IEEE Control Systems*, vol. 14, issue 1, pp. 5-12, February 1994.

[58] D. Schelle and J. Castorena, 'Buck-Converter Design Demystified', *Power Electronics Technology*, June 2006, pp. 46-53.

[59] S. P. Singh and R. S. Sutton, 'Reinforcement Learning with Replacing Eligibility Traces', *Machine Learning*, Kluwer Academic Publishers, 22, pp. 123-158, 1996.

[60] S. P. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvari, 'Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms', *Machine Learning*, vol. 38, no. 3, pp. 287-308, 2000.

[61] A. Skowron and J. Stepaniuk, 'Tolerance Approximation Spaces', *Fundamenta Informaticae*, vol. 27, no. 2/3, pp. 245-253, 1996.

[62] *Sparkfun Electronics: General PIC Tutorials*, 2005, http://www.sparkfun.com/commerce/hdr.php?p=tutorials *Last checked for content, February 1, 2007.*

[63] J. Stepaniuk, 'Approximation Spaces, Reducts and Representatives', in L. Polkowski and A. Skowron (Eds.), *Rough Sets in Knowledge Discovery 2, Studies in Fuzziness and Soft Computing*, 19, pp. 109-126, Heidelberg: Springer Verlag, 1998.

[64] C. E. Strangio, *The RS232 Standard*, http://www.camiresearch.com/ *Last checked for content, February 1, 2007.*

[65] R. S. Sutton, A. G. Barto, and C. W. Anderson, 'Neuronlike Adaptive Elements That Can Solve Difficult Learning Problems', *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp 834-847, September/October 1983.

[66] R. S. Sutton, 'Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding', *Advances in Neural Information Processing Systems*, The MIT Press, 8, pp. 1038-1044, 1996.

[67] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA: The MIT Press, 1998.

[68] R. W. Swiniarski and A. Skowron, 'Rough set methods in feature selection and recognition', *Pattern Recognition Letters 24*, pp. 833-849, 2002.

[69] *Technologic Systems: TS-5500 Users Manual*, Technologic Systems Fountain Hills, AZ, October 2003, pp. 11-12, 19, 33, 38.

[70] *Texas Instruments: TPIC0108B Datasheets*, 2002, http://focus.ti.com/docs/prod/folders/print/tpic0108b.html *Last checked for content, February 1, 2007.*

[71] N. Tinbergen, 'On aims and methods of Ethology', *Zeitschrift fur Tierpsychologie*, 20, pp. 410-433, March 1963.

[72] H. J. Trussell, E. Saber, and M. Vrhel, 'Color Image Processing: Basics and Special Issue Overview', *IEEE Signal Processing Magazine*, pp 14-22, January 2005.

[73] R. S. Villanucci, A. W. Avtgis, and W. F. Megow, *Electronic Techniques, Shop Practices and Construction, 4th edition.*, Englewood Cliffs, New Jersey: Prentice-Hall Inc, 1991.

[74] R. H. Warren, J. A. Johnson, and G. H. Huang, 'Application of Rough Sets to Environmental Engineering Models', *Transactions on Rough Sets I*, Lecture Notes in Computer Science (LNCS), vol. 3100, pp. 356-374, 2004.

[75] C. J. C. H. Watkins, 'Learning from Delayed Rewards', *Ph.D. Thesis, King's College*, University of London, UK, pp. 25-36, May 1989.

[76] C. J. C. H. Watkins and P. Dayan, 'Technical Note: Q-Learning', *Machine Learning*, 8, pp. 279-292, 1992.

[77] *Wikipedia: The PIC Microcontroller*, 2007, http://en.wikipedia.org/wiki/PIC_microcontroller *Last checked for content, February 1, 2007.*

[78] Z. M. Wojcik, 'Application of Rough Sets for Edge Enhancing Image Filters', *International Conference on Image Processing 1994*, pp. 525-529, 1994.

[79] M. Xhaard, *SPCA50X USB Camera Linux Driver*, 2004, http://sourceforge.net/ *Last checked for content, February 1, 2007.*