

Md Aman Khan

**LOCOMOTION OPTIMIZATION OF PHO-
TORESPONSIVE SMALL-SCALE RO-
BOT: A DEEP REINFORCEMENT
LEARNING APPROACH**

Faculty of Engineering and
Natural Sciences
Master of Science Thesis
November 2019

ABSTRACT

Md Aman Khan: Locomotion Optimization of Photoresponsive Small-scale Robot: A Deep Reinforcement Learning Approach
Master of Science Thesis
Tampere University
Degree Programme in Automation Engineering, MSc (Tech)
November 2019

Soft robots comprise of elastic and flexible structures, and actuatable soft materials are often used to provide stimuli-responses, remotely controlled with different kinds of external stimuli, which is beneficial for designing small-scale devices. Among different stimuli-responsive materials, liquid crystal networks (LCNs) have gained a significant amount of attention for soft small-scale robots in the past decade being stimulated and actuated by light, which is clean energy, able to transduce energy remotely, easily available and accessible to sophisticated control.

One of the persistent challenges in photoresponsive robotics is to produce controllable autonomous locomotion behavior. In this Thesis, different types of photoresponsive soft robots were used to realize light-powered locomotion, and an artificial intelligence-based approach was developed for controlling the movement. A robot tracking system, including an automatic laser steering function, was built for efficient robotic feature detection and steering the laser beam automatically to desired locations. Another robot prototype, a swimmer robot, driven by the automatically steered laser beam, showed directional movements including some degree of uncertainty and randomness in their locomotion behavior.

A novel approach is developed to deal with the challenges related to the locomotion of photoresponsive swimmer robots. Machine learning, particularly deep reinforcement learning method, was applied to develop a control policy for autonomous locomotion behavior. This method can learn from its experiences by interacting with the robot and its environment without explicit knowledge of the robot structure, constituent material, and robotic mechanics. Due to the requirement of a large number of experiences to correlate the goodness of behavior control, a simulator was developed, which mimicked the uncertain and random movement behavior of the swimmer robots. This approach effectively adapted the random movement behaviors and developed an optimal control policy to reach different destination points autonomously within a simulated environment. This work has successfully taken a step towards the autonomous locomotion control of soft photoresponsive robots.

Keywords: Soft robot, photoresponsive robot, liquid crystal network, deep reinforcement learning, deep q learning, autonomous locomotion control.

PREFACE

The research work of this thesis was conducted in Hervanta Campus of Tampere University during 2019 under a seven-month period research assistantship. My sincere gratitude to my examiner Prof. Arri Priimägi, who gave me the opportunity to work with Smart Photonic Materials (SPM) research team as a research assistant and for providing constant support, effective advice, ample resources and freedom in decision-making. He has been an inspiration for my future scientific careers. I am grateful to my supervisor and second examiner Dr. Hao Zeng for his immense cooperation and guidance during my thesis. My gratitude to the staff and colleagues at the Smart Photonic Materials group for a friendly atmosphere and assistance in practical matters. I also like to express my gratitude towards the Phototune project.

I also want to thank all of my friends for their encouragement, especially Md. Mehedy Hasan Sumon, Zahangir Khan, and M Sabbir Rahman. Last but not least, thanks to my family for supporting me along with my studies in all aspects.

Tampere, 16 November 2019

Md Aman Khan

CONTENTS

| | |
|---|----|
| 1.INTRODUCTION..... | 1 |
| 1.1 Thesis structure | 4 |
| 2.MATERIALS AND DEVICES..... | 5 |
| 2.1 Liquid crystals and liquid crystal networks..... | 5 |
| 2.2 LCN photomechanical actuation..... | 6 |
| 2.2.1 Photochemical actuation in LCNs | 7 |
| 2.2.2 Photothermal actuation in LCNs..... | 9 |
| 2.2.3 Comparison between photochemical and photothermal actuators | |
| 10 | |
| 2.3 Scaling effect on LCN robots..... | 11 |
| 2.4 LCN soft robots | 12 |
| 3.MACHINE VISION AND MACHINE LEARNING..... | 15 |
| 3.1 Machine vision | 16 |
| 3.1.1 Illumination..... | 16 |
| 3.1.2 Optical components | 17 |
| 3.1.3 Camera sensor | 17 |
| 3.1.4 Image processing..... | 18 |
| 3.2 Artificial intelligence and machine learning | 18 |
| 3.3 Reinforcement learning | 21 |
| 3.3.1 Elements of reinforcement learning..... | 22 |
| 3.3.2 Markov decision process (MDP) | 23 |
| 3.3.3 Reinforcement learning algorithms..... | 24 |
| 3.3.4 Value function | 25 |
| 3.3.5 Q-learning..... | 25 |
| 3.4 Deep reinforcement learning | 26 |
| 3.4.1 Deep Q-learning..... | 26 |
| 4.IMPLEMENTATION | 29 |
| 4.1 Laser steering workspace setup..... | 29 |
| 4.1.1 Machine vision subsystem | 29 |
| 4.1.2 Laser steering subsystem | 31 |
| 4.2 Robot tracking system..... | 33 |
| 4.2.1 Machine vision for robot tracking..... | 33 |
| 4.2.2 Communication server | 37 |
| 4.2.3 Servo motor control subsystem..... | 37 |
| 5.EXPERIMENTAL RESULTS..... | 39 |
| 5.1 Light actuation in an LCN bending strip..... | 39 |
| 5.2 Optical control in three legs walking robot..... | 41 |
| 5.3 Light propelled robotic swimmer..... | 42 |
| 5.4 Light controlled swimming | 43 |

| | |
|--|----|
| 6.REINFORCEMENT LEARNING IN LIGHT DRIVEN SWIMMING ROBOT (SIMULATIONS)..... | 46 |
| 6.1 Simulation environment..... | 46 |
| 6.1.1 Action space and observational space | 46 |
| 6.1.2 Simulator | 47 |
| 6.1.3 Reward function | 48 |
| 6.2 Setting up the agent and training..... | 49 |
| 6.3 Training without associating randomness in locomotion | 51 |
| 6.4 Training approaching the real situation..... | 53 |
| 7.CONCLUSIONS AND OUTLOOK | 59 |
| REFERENCES..... | 61 |
| APPENDIX A: SERVO MOTOR CONTROL SUBSYSTEM OBJECT | 69 |
| APPENDIX B: VEDIO ANALYSIS TOOL FOR ANGLE MEASURMENT (USER INTERFACE)..... | 73 |
| APPENDIX C: DEVELOPED DEEP Q LEARNING PROGRAM | 74 |
| APPENDIX D: DEVELOPED DEEP Q LEARNING PROGRAM FOR IMPLEMENTATION WITH MV AND LASER STEERING SYSTEM | 84 |

LIST OF FIGURES

| | | |
|------------|---|----|
| Figure 1. | Comparison of Young's moduli for different materials. Adapted with permission from Ref. [6]..... | 2 |
| Figure 2. | a. Traditional rigid-bodied robot. b. Soft-robotic gripper being driven by pneumatic actuation. c. Photoresponsive small-scale robotic gripper. b. Adapted with permission from Ref. [11], c. Adapted with permission from Ref. [12]..... | 2 |
| Figure 3. | a. LC phase to isotropic phase transition. b. Common LC phases..... | 6 |
| Figure 4. | a. Photoisomerization of azobenzene and changes in molecular geometry. b, c. Different photoinduced bending with different LC alignment, b. Homogeneously aligned, c. Homeotropically aligned. a. Adapted with permission from Ref. [37]. b, c. Adapted with permission from Ref. [38]..... | 7 |
| Figure 5. | Photothermal mechanism. Adapted with permission from Ref. [53]..... | 9 |
| Figure 6. | Relation between Artificial Intelligence, Machine Learning and Deep Learning | 19 |
| Figure 7. | Machine learning categories | 19 |
| Figure 8. | Classical control system..... | 21 |
| Figure 9. | RL in control system perspective..... | 21 |
| Figure 10. | RL framework and sub-elements..... | 22 |
| Figure 11. | The deep Q-learning algorithm with experience replay. Adapted with permission from Ref. [103]..... | 27 |
| Figure 12. | Laser steering workspace setup (front, side and rear view) and different components of the workspace | 30 |
| Figure 13. | a. An LCN robot and the robot's leg illuminated with a green laser beam (captured using a regular camera). b. Image from the machine vision system without an optical filter. c. Image from the vision system's camera with a colored glass filter..... | 31 |
| Figure 14. | Robot tracking system..... | 33 |
| Figure 15. | a. The dot grid pattern for camera calibration b. Image after calibration, including the tangential distortion correction..... | 34 |
| Figure 16. | a. State diagram for the vision system. b. Steps in the Inspect state. c. Steps under the Vision Assistant step and changes of the original image along with these steps..... | 35 |
| Figure 17. | Output of the Inspect state and rectangular boxes indicate some steps (i.e. green: ROI1, purple: ROI2 and red: matched feature)..... | 36 |
| Figure 18. | A sequence diagram of the servo motor control subsystem | 37 |
| Figure 19. | a. Timeline of LCN strip deformation dynamics during (light on at 1.5 s) and after (light off at 5.1 s) light exposure. b, c. Angle measurement, the vertical blue line is the fixed axis and the other axis follow the tip of the strip | 40 |
| Figure 20. | LCN strip bending deformation..... | 40 |
| Figure 21. | Features detection of three legs walking robot's through machine vision, photograph of the robot in the insert..... | 41 |
| Figure 22. | Locomotion timeline of a light propelled robotic swimmer..... | 43 |
| Figure 23. | Structure of a light controlled swimming robot..... | 44 |
| Figure 24. | LCN robot. a. Action states. b. Observational states | 47 |
| Figure 25. | LCN robot position simulator and visualizer..... | 48 |
| Figure 26. | Architecture of the applied deep neural network..... | 50 |
| Figure 27. | Training results without associating randomness in locomotion behavior with a fixed destination point..... | 52 |

| | | |
|------------|--|-----------|
| Figure 28. | <i>Evaluation of learned policy. a. Robot trajectory with the same target location used during training b. Robot trajectory with a different target location.....</i> | <i>52</i> |
| Figure 29. | <i>Training results with randomness in the rotational movement with a fixed destination point</i> | <i>54</i> |
| Figure 30. | <i>Evaluation of learned policy with randomness in the rotational movement. a. Robot trajectory with the same target location used during training b. Robot trajectory with a different target location</i> | <i>54</i> |
| Figure 31. | <i>Training results with randomness in the rotational movement with different destination points</i> | <i>55</i> |
| Figure 32. | <i>Evaluation of learned policy with different target locations and randomness in the rotational movement. a,b,c,d,e,f. Robot trajectory at different target location and robot successfully reaches the targets</i> | <i>55</i> |
| Figure 33. | <i>Training results with randomness in every action with different random destination points (changing the points in every 50 episodes). a, b. Conducted training in two sessions</i> | <i>56</i> |
| Figure 34. | <i>Evaluation of learned policy with different target locations and randomness factors included in all actions. At the same target, robot successfully reached using different trajectories. The same target pairs are (a,b);(c,d),(e,f);(g,h)</i> | <i>57</i> |

LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|-------|---|
| AI | Artificial Intelligence |
| CCD | Charge Coupled Device |
| CMOS | Complementary Metal Oxide Semiconductor |
| DOF | Degree of Freedom |
| DQN | Deep Q Network |
| DRL | Deep Reinforcement Learning |
| GLCNs | Glassy Liquid Crystal Networks |
| LCs | Liquid Crystals |
| LCNs | Liquid Crystal Networks |
| MV | Machine Vision |
| MDP | Markov Decision Process |
| RL | Reinforcement Learning |
| TD | Temporal Difference |

| | |
|--------|-----------------------|
| F_G | gravitational force |
| F_a | active force |
| L | characteristic length |
| μ | viscosity |
| N | normal force |
| Re | Reynold number |
| v | velocity |
| ρ | density |

1. INTRODUCTION

Robot is a device or machine that can carry out a series of pre-designed tasks, whose actions often mimic the movement of living beings, performing motions such as walking, swimming, grasping, rolling, etc. Traditionally, engineers utilize rigid materials to realize sophisticated controlled robotic systems consisting of different discrete joints and links for locomotion or manipulation. For developing more human-friendly and safe robots, scientists and engineers are exploring the capability of soft, smart and stimuli-responsive materials for designing soft-bodied machines to achieve specific robotic control [1].

Stimuli-responsive materials have the ability to react to different external stimuli, converting the input (stimulus) energy into the change of their physical and chemical properties [2]. The stimuli can be in diverse forms, such as chemicals [3], electrical or magnetic field [4], mechanical stress [5], temperature [6], humidity [7], light [8] and so on, the stimuli-responsive-materials often being called as smart or intelligent materials [9]. A wide range of stimuli-responsive materials are available in the literature for soft robotic realization, such as liquid crystal networks, shape memory polymers, hydrogels, electro- and magnetorheological fluids, and many more [6]. Among those, liquid crystal networks (LCNs) have gained an increasing amount of attention in the past decade. These soft and smart materials can utilize different stimuli such as light [10], humidity [7], electric field, heat and chemical reaction [6], to create deformation or locomotion. Particularly, LCN materials have become attractive because of the possibility of being stimulated and actuated by light, ability to transduce energy remotely as well as light is easily available and provides sophisticated control.

Depending on the rigidity of constituent materials, robots can be divided into hard-bodied robots and soft-bodied ones. A hard-bodied robot has rigid components with a limited number of links and joints. Usually, it has a very limited degree of freedom (DOF) and degree of movement. Conversely, soft robots comprise of soft actuatable materials, with Young's modulus up to several gigapascals (Figure 1) [6], flexible joints and links and can provide much more degrees of freedom for movement. Besides, the links and joints of a soft robot can have different stiffness and can be arranged in a serial or parallel fashion for creating complex movement.

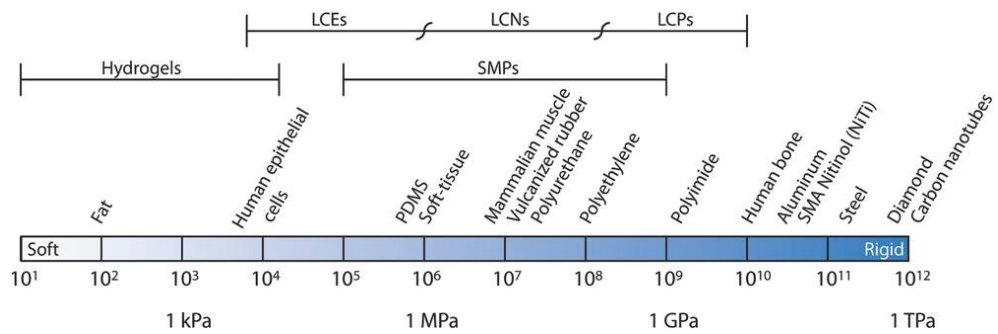


Figure 1. Comparison of Young's moduli for different materials. Adapted with permission from Ref. [6]

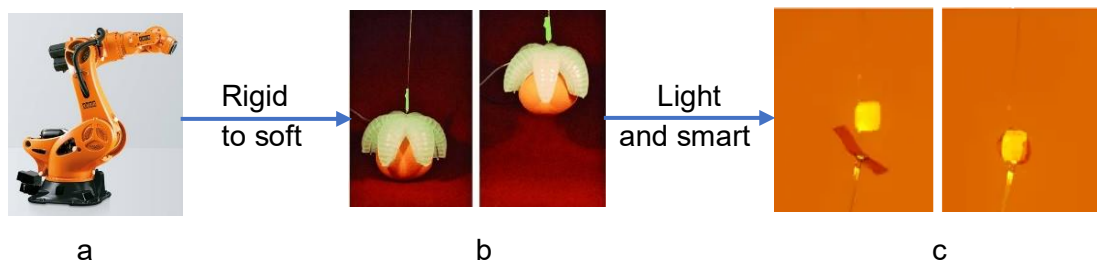


Figure 2. a. Traditional rigid-bodied robot. b. Soft-robotic gripper being driven by pneumatic actuation. c. Photoresponsive small-scale robotic gripper. b. Adapted with permission from Ref. [11], c. Adapted with permission from Ref. [12]

Soft robots have several advantages over hard-bodied robots, such as adaptation to unpredictable obstacles, continuous actuation and shape change, etc. Typical examples are shown in Figure 2, where a silicon-based soft pneumatic hand can grip an egg by adapting its shape and curvature, which enables an easy handle of fragile objects without precise machine programming (Figure 2b). In contrast, rigid gripper needs explicit programming in each moving step to reach the target and accurate control of the force for a precise grasping (Figure 2a) [13]. Nowadays, most of the soft robots use pneumatic actuation based on air or liquid tube powering. Designing small-scale robots for accomplishing different tasks at small length scales is extremely difficult using pneumatic tube connection. For harnessing full potential and achieving practical applications, small-scale soft robots need actuation, control, and power storage systems embedded into one soft body, which raises the importance of stimuli-responsive smart materials [1][6][13].

Many reports have shown that soft smart materials are becoming a great candidate for small-scale robotics, one pioneering example being shown in Figure 2c, where a photoresponsive gripper can grab a falling object based on the light reflectance from the object [12]. Smart materials can be actuated remotely with different stimuli. In this sense, the power source can be separated from the robot body and deliver remotely the necessary energy [10]. As demonstrated by the example shown in Figure 2, photoresponsive

materials can perform shape changes, and generally this kind of actuation can be versatile due to the alignment control technique. To obtain the actuation, photo-sensitive elements like photoswitches (molecular motors) are often used, in which the cooperative movements can bring up deformation from a molecular scale to a macroscopic level [14]. Traditional hard-bodied robot subsystems consist of, for example, actuation system, sensing elements, controller, computational system and power [1]. The photoresponsive soft robot can have most of these subsystems into one monolithic sample: photoactuation serves as an actuation plus a power delivery system, pre-programmed actuator (e.g. through photopatterning) serves as a control/sensing system. In this aspect, photoreponsive materials can enable not only the miniaturization, but also advantageous robotic functions based on specific material response design.

One of the persistent challenges in soft robotics is to design controllable bodies for delivering desirable behaviors. The traditional strategy for controlling a hard-bodied robot is based on manipulating a series of rigid joints, each representing six degrees of freedom of movement. This strategy is not suitable for soft robots because of the fact that soft materials are flexible and often exhibit high degrees of freedom, such as twisting, bending, wrinkling, etc., thus presenting a large number of DOF. Some theoretical models have been developed to describe bending in soft matter [1][15]. It remains a great challenge to develop an accurate model for predicting machine performance or efficient strategy to execute tasks due to soft material properties [15]–[17]. Still, there is no well-developed model or reliable algorithm for soft robotic movement. Thus, new strategies and approaches are needed for the control of soft robots.

In this thesis, a novel machine-learning-based approach is developed to tackle the challenges related to the locomotion behavior of photoresponsive robots. A robotic function can be split into perception task which is related to acquiring essential information from the environment and control task which is related to achieve a goal based on that information [18].

Machine vision system offers an efficient way of detecting small scale robots, including information about their different parts and surrounding environment. Besides these photoresponsive robots are driven by light, and each part of the robot can be actuated upon a laser or an LED light field for creating robotic movement. The laser is preferable for pin-point excitation of different robot section. A control system allows steering the laser beam to desired locations automatically. A robot tracking system is developed for efficient detection of photoresponsive robots, meanwhile synchronized with the control sys-

tem to drive the laser beam to the desired parts, according to developed algorithm. Different types of small-scale robots, such as a bending cantilever arm, walking robot and floating swimmer, are tested under these robot vision/control/tracking systems.

Currently, reinforcement learning is used in different applications offering excellent decision-making capabilities [19]. In soft robot control, reinforcement learning can play a significant role because it can produce an optimized control policy from its experiences that obtained by interaction between robot and the environment, without any explicit knowledge of the material, robot structure or robotic mechanism [20][21]. DeepMind team of Google successfully utilized a conventional reinforcement learning called Q-learning with deep neural networks to play computer games like Atari [22]. After that, this method is adopted in many applications such as in mobile robot path planning and autonomous navigation [23]–[26], autonomous driving [19][27], robot motion control[28][29] and many more.

To develop an effective and optimal control policy for locomotion behavior of photoreponsive robots, a deep reinforcement learning method is applied. This technique requires a large amount of experience to correlate the goodness of control of the robot. Thus, the methods are only applied in a computer simulation environment for this Thesis study. The simulated model contains specific operations, which mimic the uncertainty and random movement behavior of a photoresponsive swimmer robot. This study serves as a primary trial to develop effective adaptation in control of soft robotic movement and optimal control policy for future robots.

1.1 Thesis structure

The thesis is divided into 7 chapters. After the Introduction, Chapter 2 introduces a concise concept of liquid crystals (LCs) and liquid crystal networks (LCNs), photoactuation, and effects of different forces on small scale robot as well as reported LCN robots. Chapter 3 presents an overview of machine vision system and relevant concepts of reinforcement learning, which are used in this thesis study. Chapter 4 describes the experimental workspace setup and the implementation of robot tracking system. Chapter 5 highlights the experimental results with different types of light-driven robots. Chapter 6 proceeds with describing the RL environment, including the developed simulator, agent and training parameters, and the training results along with the robot's autonomous movement to evaluate the learned control policy. Finally, Chapter 7 summarizes the overall outcomes, in addition to providing future perspectives and outlining the potential of this research.

2. MATERIALS AND DEVICES

Smart stimuli-responsive materials are capable of being actuated remotely by using different kinds of stimuli, which is beneficial for designing small scale robots. liquid crystal networks (LCNs) are soft, smart, and stimuli-responsive material. LCNs have the ability to transduce light energy into mechanical work output, which is one of most attractive features for devising small wireless devices. In this chapter, firstly the concise concepts of liquid crystals and liquid crystal networks are presented. Then photomechanics in LCNs, including photothermal and photochemical actuation modes are described. Finally, effects of different forces on small scale LCN robots and various kinds of LCN robots and their locomotion capabilities are briefly discussed.

2.1 Liquid crystals and liquid crystal networks

Liquid crystals (LCs) are special state of matter [30][31]. This state is an intermediary between crystalline solids and isotropic liquids, also known as mesophase. LCs can retain anisotropy (positional and orientational) characteristic of crystalline solids, along with the fluidic property of liquids [7][31][32]. LCs undergo transition from anisotropic LC phase (order) to isotropic phase (disorder) upon heating (Figure 3a) [10].

LCs are divided into thermotropic and lyotropic categories. LCs phase occurs for the former type at a particular range of temperature, whereas the latter one requires a particular concentration in solution besides temperature for achieving the LC phase. Thermotropic LCs have different subphases based on positional and orientational alignment, such as nematic, smectic and cholesteric phases (Figure 3b). In nematic phase, the molecules have an average direction (represented by a vector called director) in their orientational order but no positional order. Smectic phase has positional and orientational molecular order. Cholesteric phase is like nematic phase though the molecules are arranged in helical orientation along the director [7].

Liquid crystal networks are crosslinked synthetic polymer systems [15][16]. This unique solid and can be fabricated in different ways, allowing to retain the molecular alignment order of LCs even in the solid state [34]. They combine the liquid crystals' anisotropy properties and the mechanical properties of polymers. LCNs can be classified into two types depending on the crosslinking, namely liquid crystal elastomers (LCEs) (weakly crosslinked), whose glass-transition temperature, T_g , is below room temperature and modulus is approximately between 0.1 to 5 MPa, and glassy liquid crystal Networks

(GLCNs) (moderate or densely crosslinked) whose T_g is approximately between 40°C to 120°C and modulus is approximately between 0.8 to 2 GPa [15][16].

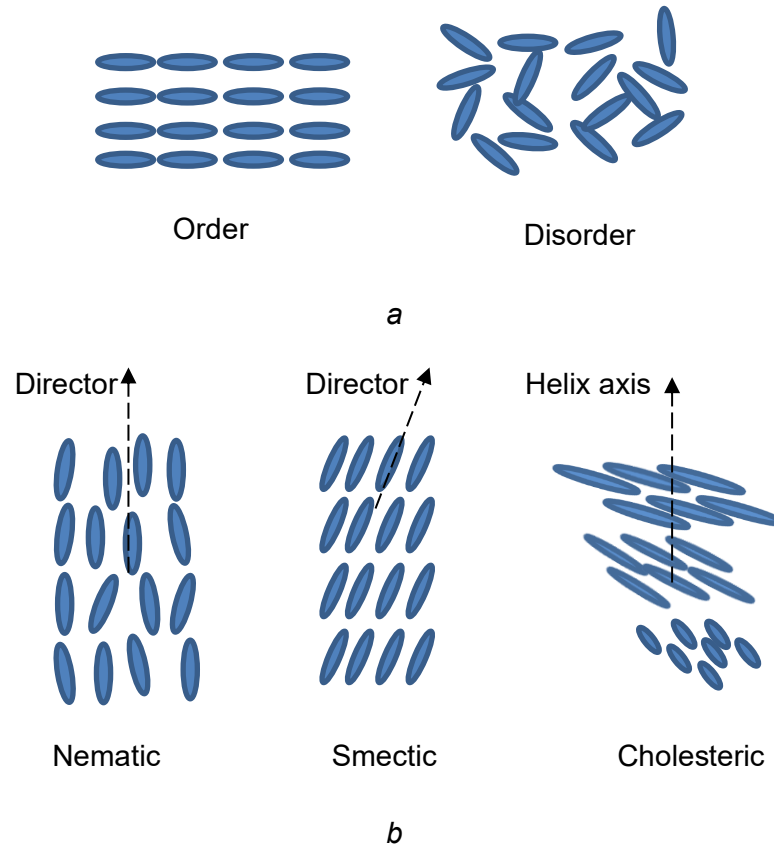


Figure 3. *a. LC phase to isotropic phase transition. b. Common LC phases*

2.2 LCN photomechanical actuation

Actuators are devices that transduce other forms of energy into mechanical work. LCNs are stimuli-responsive materials [35]. LCNs hold the anisotropy properties of LCs, and the response of the material system is amplified from molecular level to macroscopic scale by collective interaction between LC molecules and polymer networks. Hence, LCNs serve a good basis for macroscopic actuation. LCNs can, in principle, respond to thermal, electrical, and optical stimuli, hence transducing these forms of energy into mechanical motion such as deflection, deformation, or other types of motion [35]. Among these stimuli, light is more feasible because of being clean, remotely and precisely controllable. Azobenzene derivatives are used as photoswitch to design photoresponsive macroscopic actuator. Photoswitches construct the molecular level deformable system (molecular-scale motion) and their cooperative motion through the network can be

amplified to create macroscopic shape-change in LCNs, which results in photomechanical actuation [7][31]. This actuation in LCNs incorporates two mechanisms: photothermal and photochemical actuation [7].

2.2.1 Photochemical actuation in LCNs

Photochemical actuation is driven by the reversible photoisomerization between two isomeric states in photoswitchable molecules [7]. For LCNs, azobenzene derivatives are the most popular photoswitches [7][36]. Azobenzene is an aromatic molecule composed of two phenyl rings linked by an azo group (N=N). Unsubstituted azobenzene and its derivatives go through reversible photoisomerization, which occurs around the double bond of an azo functional group (N=N) by switching between two states, *i.e.* a stable trans-form (E isomeric state) and metastable cis-form (Z isomeric state) (Figure 4a) [37]. Priimagi and coworkers highlighted some attractive features of azobenzene derivatives as photoswitches in LCNs: comparatively easy synthesis of a wide range of azobenzene derivatives with different activation wavelengths and photochemical properties, the miscible property of trans-azobenzene with multiple LCs and destabilization of LC phase due to angular shape of cis-azobenzene [13].

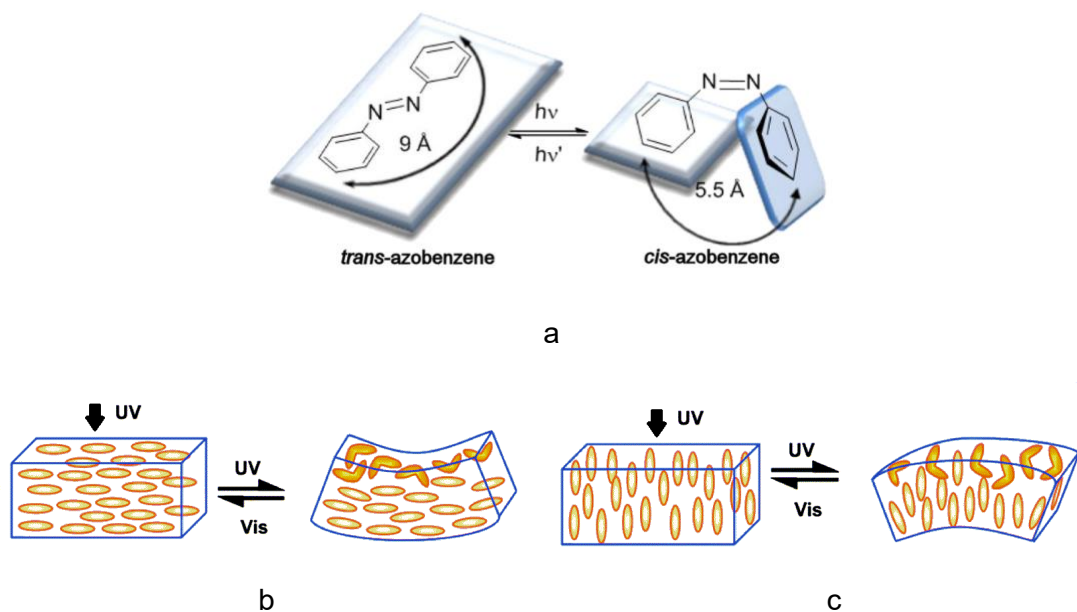


Figure 4. *a.* Photoisomerization of azobenzene and changes in molecular geometry. *b, c.* Different photoinduced bending with different LC alignment, *b.* Homogeneously aligned, *c.* Homeotropically aligned. *a.* Adapted with permission from Ref. [37]. *b, c.* Adapted with permission from Ref. [38]

Trans-to-cis isomerization occurs when trans isomer of azobenzene (unsubstituted azobenzene and its derivatives) is irradiated with UV light (typically 320-380 nm). The isomerization is a reversible reaction, and cis-to-trans isomerization occurs thermally or upon

visible-light irradiation. Trans-to-cis and cis-to-trans isomerization of azobenzene exhibit different changes in physical properties such as molecular geometry, dipole moment or absorption spectrum [37][39]. Molecular geometry of trans-azobenzene is planar with 9.0 Å distance between the 4 and 4' positions of the phenyl rings. On the other hand, cis-azobenzene is more globular, particularly phenyl rings of cis-azobenzene are twisted at 90° relative to the C–N=N–C plane, which reduces the distance between the 4 and 4' positions of phenyl rings to 5.5 Å. The geometric changes also lead to a change in the dipole moment, *i.e.*, no dipole moment in trans-form of parent azobenzene and 3.1 D dipole moment in cis-form [7][37][40]. Molecular-level photoisomerization efficiently modulates molecular order within the LCN polymer network, inducing LC-to-isotropic phase transition, and triggering photomechanical actuation in macroscopic free-standing samples [13][36][41].

The lifetime of cis-azobenzene is an important factor, since it determines the stability of the photodeformed state [13]. Chemical substitution plays a significant role in the control of the cis-lifetime and tuning the wavelength for activating trans-to-cis reaction [13]. For example, the cis-lifetime of tautomerizable push-pull azo derivatives can be less than a millisecond [42], whereas the ones in heterocyclic or ortho-substituted azobenzenes can last for months or years [43][44]. Supitchaya and coworkers present bi-stable photoactuators containing fluorinated azobenzenes, which can preserve the photochemically deformed shape for several days [45]. Regarding the activation wavelength, some conceptual strategies have been proposed to induce trans-to-cis reaction upon visible illumination, aiming for a more efficient solar energy harvesting and human-friendly interaction [13][45][46].

Photochemical actuators often create photoinduced bending whose direction depends on the specific molecular alignment. LCN actuators with homogeneous alignment exhibit bending towards the light source due to contraction of the LCN surface along the director (Figure 4b). On the contrary, LCN actuators with homeotropic alignment exhibit bending in the opposite direction because of the expansion of the light exposed surface (Figure 4c) [38]. LCN actuator with splayed alignment across thickness can produce different strain within a single monolithic layer through forming expansion and contraction on opposite sides [12]. This alignment pattern yields efficient and noticeable bending deformation. Also this alignment pattern defines the bending direction and axis of the actuator irrespective of incident light [47]. These features are attractive for devising robots [12].

Azobenzene moieties have strong absorption properties that restrict light penetration into the bulk. As a result, photoisomerization is limited to the LCN surface [13][19]. Upon

illumination, the concentration of cis-azobenzene varies with the distance from the surface, which generates a non-uniform stress distribution across the thickness and produces bending in the LCN sample.

2.2.2 Photothermal actuation in LCNs

Photoactuation in LCNs can also be triggered through photothermal mechanism. For this mechanism, photosensitive moieties, e.g. organic dyes or nanoparticles are introduced in LCN as nanoscopic heat generators that absorb photons. Non-radiative thermal processes convert the energy from the absorbed photons into heat, triggers order-disorder transition of the LCNs and yielding macroscopic deformation of the entire network (Figure 5) [5][15][30]. Azobenzenes, which have short cis-lifetime or concurrent trans-cis and cis-trans activation, are efficient photothermal heat generators. Thus, several sophisticated photoactuator demonstrations achieved recently are designed by utilizing this actuation mechanism [11][22][50]. Interestingly, photostabilizers and organic dyes, which are not even photoisomerizable, can be used to induce photothermal heating in LCNs [51]. Though the key issue is the solubility of dopants into LC mixture, most of the time a small amount of dopants (ca. 1 %) is sufficient for a significant photothermal heat generation [51]. By using a suitable dye, photothermal actuation can be triggered by a large range in spectrum, such as visible-near infrared wavelengths [52].

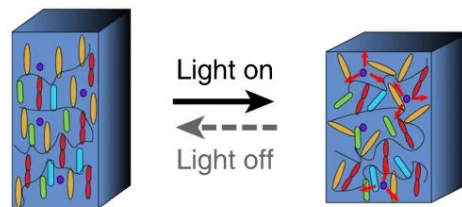


Figure 5. *Photothermal mechanism. Adapted with permission from Ref. [53]*

Nanoparticles of inorganic carbon or metals also can be doped into LCNs for photothermal actuation [54]. Carbon nanotubes able to align along the LC mesogen direction. Gold nanoparticles of different sizes and shapes, e.g., rod, stars, govern the plasmon resonance wavelength which determines the photothermal absorption. Like organic dyes, a small amount of inorganic nanoparticles is needed for photothermal actuation, however the major problem is the poor miscibility, which causes inferior mechanical properties, enhanced light scattering and deficient performance of photothermal actuation [11][26]. Though suitable surface functionalization of these particles can be adopted to minimize these problems [25], organic dyes are more appropriate for diverse applications.

2.2.3 Comparison between photochemical and photothermal actuators

For devising actuators and small-scale robots, both photochemical and photothermal mechanisms are significant. Depending upon the chemical characteristics and the physical features of these mechanisms, different distinct use cases for light-responsive robots can be figured out [13]. The differences between these mechanisms are discussed below.

Based on chemical characteristics, the critical difference between photochemical and photothermal actuators is the position of active units within the LCN. Azobenzenes are the active units in photochemically activated LCN within the LCN polymer to yield efficient actuation. Whereas, dyes or nanoparticle are the active units in photothermally activated LCN which are doped into the system, without any need to be crosslinked with the polymer chains [10]. Another difference is the activation wavelength. For photochemical actuator, typically trans-to-cis isomerization is triggered by UV or deep blue light, and reverse isomerization is occurred by irradiating with 450-550 nm. Researchers are trying to shift the activation wavelength towards green or near-infrared wavelength (using nanoparticles) [55]–[57]. On the other side, photothermal actuation is at the best at the wavelength the dyes absorb, leading human-friendly visible or near-infrared light activation [10]. Finally, changing the photoactive unit concentration may lead to different results. Increasing the concentration of azobenzene in photochemical LCN assists the actuation due to the enhanced absorption gradient [10], as presented by the group of Ikeda [58][59]. On the contrary, a photothermal actuator with a small amount of dye can absorb the major portion of incoming photons, and produce rapid actuation in a reversible way [60].

Also, different physical features can be distinguished between photochemical and photothermal actuators, such as actuation speed, suitable environment to actuate, absorption gradient. In photochemical actuation, the molecular level photoisomerization happened immediately upon irradiation, but the macroscopic deformation of LCN requires seconds or minutes [36], *i.e.* slow response. On the other side, long cis-lifetime gives the actuator a bi-stable feature [61]; in other words, the deformed shape can be retained after ceasing the excitation. Photothermal actuators show rapid deformation which can take only milliseconds or seconds [10]. Also, fast recovery of original shape occurs when ceasing excitation, and cooling down the actuator. Photochemical actuators have a distinct advantage of functioning in different environments including in aqueous medium [62][63]. Whereas photothermal mechanism is hindered in an aqueous medium because it relies on heat conduction by raising the temperature within the actuator. But the higher

heat conductivity of a liquid (e.g. water is 20 times of the one in air), reduces the actuator temperature, or a higher order of magnitude light intensity is required to induce the equivalent amount of deformation in water compared to the one operated in the air [10]. Besides, as the photochemical mechanism depends on absorption gradient, it can yield bending, but in-plane actuation is challenging. Unlike photochemical actuator, photothermal actuation in LCN can yield both in-plane actuation and out-of-plane bending [7] [10].

2.3 Scaling effect on LCN robots

From big-sized machines to microrobots, it requires geometric change in the material driven by through actuation. For designing a photoresponsive robot or device and yielding controlled locomotion or function, the interaction between the robot and its working environment is essential to investigate [13][64].

Though the physics for all objects, either at the microscopic or macroscopic scale, is the same, different forces and their contributions change depending on the scale of the object. In other words, the ratio between forces as well as related physical phenomena may change with the dimension of the device. Therefore, different forces may start dominating the object motion depending on the object characteristic length (L), known as scaling effect [65]. Here, L^s notation is used to represent the dimension influences where s is the scaling factor. For instance, the gravitational force, F_G and inertia are both related to the volume of the object, which scales as L^3 . In LCN robot, the photo-induced elastic force is the active force F_a , related to the cross-section of the material, thus scaling as L^2 . The overall force ratio ($F_a : F_G$) scales as L^{-1} , representing a strong size dependency. The ratio between F_a and F_G is comparatively high for a small LCN micro-robot compared to ones with bigger size. This kind of difference at different scales implies the requirement of different design concepts and strategies to achieve efficient locomotion. Interestingly, hints are given by nature already: big mammals have strong skeletons for supporting their weight, meanwhile they run using two or four legs when inertia plays a significant role. However, for small insects, gravitation plays a minor role and they can easily jump a long distance. In many cases, the insects grow specific hairy architecture on their skins to prevent adhesion, whose effect is typically enhanced at microscale [66].

The locomotion of a microrobot at an interface contemplates as walking where adhesion and friction forces arise mostly. This adhesion is the combination of van der Waals, capillary, and other forces. Van der Waals forces appear within a few nanometres due to the fluctuating dipole in the material, which strongly depends on the contact area and surface roughness. Capillary forces are enhanced in the presence of liquid in between two solid surfaces, where the liquid tends to reduce the surface energy by minimization

of the surface area, thus posing an attracting force [67]. The classical equation for friction is:

$$f = \mu \times N$$

where μ = friction coefficient and N = normal force [68]. This equation does not properly describe soft materials experiencing adhesion. Particularly, in micro-world, gravitation is negligible and friction/adhesion dominate, which is more unpredictable and depends on the material softness and surface conditions [68]. When LCN robot is photo-heated, it decreases the rigidity which results in a more dynamic adhesion/friction force. Due to this reason, the precise control of LCN locomotion is challenging even though the cyclic shape changes are predictable. Different methods have been introduced for LCN walking robot to reduce the overall friction forces and set up reasonable amount of friction bias to promote the directional walking tendency, such as adding an extra leg of rigid materials, using the conical tip to minimize the effect contact area [67] [68].

The locomotion of a microrobot in a homogeneous liquid medium contemplates as swimming. The interaction between microrobot and the medium is related to Reynold number, which is the ratio of the internal forces and viscous forces [69]:

$$Re = \rho v d / \mu;$$

where, ρ = density of the liquid, μ = viscosity of the liquid, v = velocity of the object, and d = characteristic size of the object. It is easy to notice that the Reynold number scales as L^2 . Typical example in nature are: a gigantic whale can have Re of 10^7 , fish about 1-10, and bacteria of 10^{-4} [70]. For large Re , swimming locomotion is dominated by inertia forces whereas for small Re , locomotion is dominated by viscous forces. If a microrobot has low Re ($\ll 1$), the Stokes equation shows that a perfect time reciprocal motion is unable to produce net motion [69]. Therefore, such swimmer requires a specific actuation sequence to yield effective locomotion.

LCN robots are light actuated *i.e.* powered and controlled by light energy, which ranges from tens of milliwatts per square centimetre (led source) to a hundred watts per centimetres (focused laser beam), depending on their scales. Photothermal heating speed of an LCN robot is based on the heat capacity (scales as L^3). In this sense, decrease of size results in rapidly increase in photothermal actuation speed [13].

2.4 LCN soft robots

Photoresponsive LCN robot only utilizes light energy to produce elastic forces inside the material that enables the robot to overcome the resistance of the surrounding medium

[10][71]. This kind of dynamics can be found in nature, which serves as a source of inspiration in designing and devising LCN robots. From the past decades, substantial achievements have been done in devising LCN robots such as walkers, swimmers robots, etc., but still, the example of multidirectional locomotion is rare and no autonomous locomotion control policy has been reported.

The first photoresponsive robotic function was introduced by Ikeda and coworkers in 2008 [13]. They present a plastic motor consisting of a pulley system where an LCN strip is used as a belt. It can produce rotation upon UV and visible light radiation in a sequence, hence the light is able to be transduced into mechanical energy [72]. After that year, they introduced the first LCN based walker (like an inchworm) and a flexible robotic arm showing multi-degree of movement [73]. The inchworm like walker was able to walk only in one direction under alternative UV and visible light illumination [73]. Another inchworm like robot is presented by Kohlmeyer *et al.* which can be actuated by infrared light and is able to crawl up a 50° inclined ratcheted substrate [74]. Zeng *et al.* report a microscopic walker that is able to rotate, walk and jump depending on the surface condition [75]. These robotic motions rely on spatial and temporal control of light, *i.e.* switching (on or off) of the light source and scanning of a light beam. This control of light pattern induces cyclic shape changes in the pre-patterned LCN, together with the interaction between robot and contact surface (accounting the friction bias, etc.), which is essential for locomotion [13]. Utilizing the spatial modulation of light, Wasylczyk and coworkers showed a caterpillar-like walking robot capable of moving forward and backward [76]. Gelebart *et al.* demonstrated an LCN film that also moved forward by wave generation under constant light field [55]. White and coworkers presented a spiral ribbon of LCN, which was able to roll over a long distance in a particular direction without the use of either temporal or spatial control of a static illumination [54].

Huang *et al.* designed a robot equipped with an LCN strip and confined inside a glass tube, which swam using temporal control of light [77]. The glass tube worked as a guide to move in one direction. In 2016, Palagi *et al.* presented a back and forth swimming locomotion of a cylindrical microrobot by creating traveling waves in the material with structured light [78]. For the first time, they presented in plane-controlled swimming movement using a disk-shaped microrobot by producing traveling wave [78]. Zeng *et al.* realized an LCN walker robot mimicking caterpillar larva on paper surface and blazed grating [79].

LCNs robots have advantages over traditional rigid bodied robots at small-scale and in the aspect of human-friendly applications. In the last two decades, extensive research is going on LCNs and LCN robots, and the field of LCN robot research is are evolving

rapidly. Different robotic functions are already demonstrated in different environments and upon different illumination conditions. However, the control of locomotion in existing robots still lacks reliable strategy, and it remains a great challenge to fully understand the locomotive mechanism in soft matters. Though, pre-programmable and reversible shape change and photoresponsive features of LCNs pave the way to a new class of soft robotics, optimization of the locomotion in all these LCN based devices play significant role in further research.

3. MACHINE VISION AND MACHINE LEARNING

The specific tasks to deliver robotic functions can be divided into perception and control tasks. Perception task is related to acquiring required information from the environment through sensors. Control task is to accomplish a goal based on the acquired information [18]. While dealing with perception task, machine vision is often used as an efficient system that gathers required information [80], and control task requires a control strategy to utilize the information received from the machine vision system to achieve robotic function.

Soft robotics is becoming a fast-developing research field in the past few years, crossing different disciplines such as robotics, materials science, biotechnology, optics and machine learning [16]. Soft robots have impressive features like large degree of freedom in actuation enabling bending, coiling or twisting, etc., possibility of grabbing objects of irregular shapes and potential to use in human-friendly applications [16][81]. However, it remains a great challenge to develop an accurate model for predicting machine performance or efficient strategy to execute tasks due to soft material properties [15][16][17]. Currently, no general model can provide a compressive analysis of the dynamics and kinematics of soft robots.

Extensive research has been done on systematic automation, particularly in vehicles, drones, *i.e.*, rigid machines in tough environments [24]. For autonomous operation in a complex and dynamic environment, the robot requires a rational decision-making process to take a suitable reaction based on the available information. In most cases, robotic system has very limited information accessible from the environment, and the uncertainty of the environment condition further increases the level of difficulty [25]. Soft robots possess multiple degrees of freedom, dynamic mechanical properties, thus special automation strategy must be developed to fit the situation in soft robotics. Reinforcement learning, a subdivision of machine learning, has an excellent decision-making ability without requiring knowledge of robot inside structure or constituted material [18][19][21]. By using this method, an optimized control policy can be expected for the use in controlling the soft robot locomotion.

This chapter firstly presents a brief overview of machine vision and its subsystems. Then, artificial intelligence, machine learning, deep learning, and their mutual relationships are introduced. Specific attention will be focused on reinforcement learning, its elements, and different related methods. Finally, deep reinforcement learning will be discussed.

3.1 Machine vision

Machine vision (MV) is a branch of systems engineered with mechanical, optical, electronic, and software systems, which try to encompass the science and engineering of vision studies [80][82]. MV extracts useful information by efficient detection and verification [80], thus has become popular in industrial manufacturing, and the extended applications in fields such as robotics, face recognition systems (or fingerprint), etc. [80]. A typical MV system consists of illumination, optical components, camera sensor, image processing and software.

3.1.1 Illumination

Illumination is an integral part of a machine vision system. The objective of illumination is to generate a vision through machine's "eyes" – make desirable features from the target visible and unwanted features being suppressed. Different illumination methods can be used to interact with the object and receiving the feedback information to achieve such the objective [80][83][84]. For instance, depending on the directional properties of light source illumination, one can use: (i) diffuse illumination, where the light is emitted by a source in all directions evenly or (ii) directional illumination, where the light source emits light in specific directions. For a special case, when the source emits light rays parallelly along one single direction, it is called telecentric illumination[85].

Depending on the relative position of light source and camera, illumination can be considered as: (i) front light illumination – light source and camera are kept on the same side with respect to the object, and (ii) back light illumination – light source and camera are placed on the opposite side. Depending on the incident angle of the light source, front light illumination can be divided into (i) bright field and (ii) dark field. In dark field illumination, light source is mounted at a small angle to the surface of the object, whereas the opposite for bright field illumination [80][83]. Table 1. shows some of the most popular methods combining these categories and their advantages for machine vision [85],

Table 1. *Most popular methods and their advantages for machine vision*

| Methods | Potentials |
|--|--|
| Diffuse bright field front light illumination | Prevent shadows and reduce reflections |
| Directed bright field front light illumination | Create shadows in cavities |
| Directed dark field front light illumination | Enhance indentation and protrusion features, visibility for texture, and engrave patterns. |
| Diffuse dark field backlight illumination | Detect contours |

3.1.2 Optical components

Optical components are a subsystem dealing with image acquisition, by which useful optical signal is collected and noise reduced before being detected by the camera sensor. The optics is crucial for an efficient repetitive task, and it assists to produce good quality image and reduces the effort in digital image processing. As a result, the whole vision system can become faster and more reliable. Depending on the system requirements, optical subsystem consists of different components such as filters, lenses, absorbing background, etc. [80].

A filter may enhance image contrast by blocking undesirable wavelengths. Coated interference filters and colored glass filters are the two most often used filters. Coated interference filter has a particular blocking and transmittance range, *i.e.* well-defined spectral band. They can be bandpass, bandstop, shortpass, longpass and notch filters. Colored glass filters are manufactured by adding a dopant element in the glass, which is responsible for altering the absorption and transmission spectra. Comparatively, colored filter is most prevalent in vision system application because of low cost. Moreover, colored filter does not shift the wavelength transmission and makes no change in the spectral properties despite a change in angle with respect to the optical axis [80][86].

Different types of lenses can be used to acquire the image, such as fixed focal lens, zoom lens, and telecentric lens. Selecting a suitable lens involves consideration of the factors of field of view, focal length, depth of view, camera sensor size, etc. Fixed focal lens has a fixed angular field of view. It is ideal when the distance from the target does not change. Zoom lens operates over a wide range of focal lengths and is suitable when a change of field of view becomes necessary during the operation. Unlike the other two, telecentric lens has no angular component to the field of view, and a constant field of view at any distance from the object, thus its magnification remains unchanged [87].

3.1.3 Camera sensor

The core objective of a camera is to create images when the camera sensor is illuminated by light that is collected by the lens. Camera sensors are solid-state electronic devices consisting of photodetecting pixels. The sensor size and electronic readout format are essential properties of a camera. Typically, digital camera sensor technology can be categorized into Charge-Coupled Device (CCD) and Complementary Metal Oxide Semiconductor (CMOS) [88][89]. CCD sensor is a silicon chip with an array of photosensitive sites that convert light into charges. The charges are moved into a serial readout resistor through transfer gates and converted into voltages. After amplifying

those voltages, analog to digital converter is used to produce digital pixel information [89]. On the other hand, serial readout register is not required in a CMOS sensor. The photodetector sites convert charges into voltages directly, *i.e.* the analog to digital processing is done in pixel level, and a row-column select circuit is used for readout [88][89]. CMOS has become a more popular sensor over CCD because of many advantages such as smaller pixel size, better low-light performance, lower dark noise, higher fidelity image and dynamic range, lower power consumption and lower cost in manufacture [2][14]–[16].

3.1.4 Image processing

After capturing an image, the next step is to analyze it, which includes preprocessing to enhance the image quality, camera calibration for accurate measurement, and different algorithms for extracting desirable features of the targeted object. Image smoothing techniques are used for reducing the noise in order to enhance the image quality by the application of Gaussian filter, mean filter, linear filter, etc., algorithmically [80][91][92]. For detecting a particular type of object, algorithms can be developed by incorporating many basic techniques such as geometric transformations, image segmentation, feature extraction, edge extraction, fitting geometric primitives, etc. [80][91]. However, these methods are not robust because different algorithms are required for finding different types of objects. For a practical application, template matching methods are widely used since they can detect objects by using a prototype, yielding easiness of finding different objects. Besides, standard software packages are available for utilizing template matching methods' functionalities. Different types of template matching methods have been developed, such as gray value-based matching, matching using image pyramids, robust template matching, matching geometric primitives and shape-based matching [80][85].

3.2 Artificial intelligence and machine learning

Artificial intelligence (AI) is a general term to indicate the utilization of computers to model intelligent behaviors without any help, or with minimum help, from humans, thus artificially mimicking human intelligence [93]. AI is successfully implemented in different disciplines such as speech recognition, image recognition, cancer cell detection, etc. [94]. The relation between artificial intelligence, machine learning, and deep learning can be seen from the subset structure shown in Figure 6 [95].

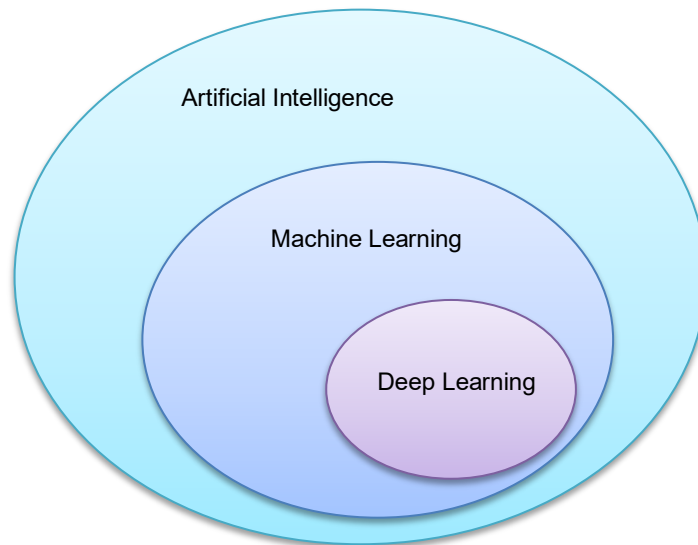


Figure 6. *Relation between Artificial Intelligence, Machine Learning and Deep Learning*

Machine learning is a type of algorithm that enables a computer to learn without explicit programming [96]. According to Dr. Tom M. Mitchell, “A *computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E* ” [97]. In other words, it is a program that can automatically learn from its experience, *i.e.* from the input data [95][97]. Machine learning algorithms can be divided into three broad categories (Figure 7) based on the data receiving method and the manner of giving feedback on the learning process. They are supervised learning, unsupervised learning and reinforcement learning.

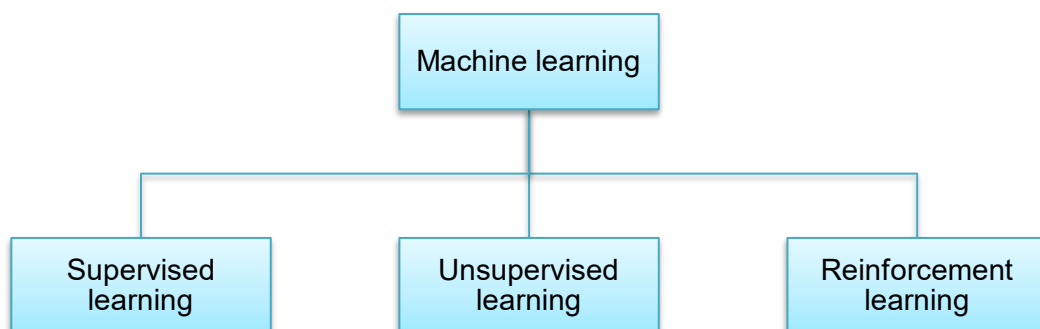


Figure 7. *Machine learning categories*

Supervised learning is nowadays the most common used method in machine learning. Supervised learning algorithms use training data sets, which are correctly labeled be-

forehand, *i.e.* the correct answer is already known during the training sessions. In training, the algorithm predicts the results and tries to find the correlation between the data and the results. The aim is to develop the mapping function at an optimum level of performance that can correctly predict the response of a new unseen instance [98]. Supervised learning is successfully used for solving real-world problems, such as recognition of speech, handwriting, face patterns, natural language processing, bank credit scoring, medical imaging, etc. [99].

Unsupervised learning algorithms discover a structure or pattern through common elements of an unlabeled data set. Thus, these algorithms do not require a training dataset. As the data is not sorted or classified beforehand, these algorithms are more complex and processing-time intensive than supervised learning [98][100]. Credit card fraud detection, market analysis, fault detection, cancer cell detection, gene sequence study, anomaly detection in a long series of data are among the various applications of unsupervised learning [99].

Reinforcement learning (RL) algorithms are real-time learning algorithms. Different from supervised and unsupervised learning which are based on datasets acquired beforehand, RL is an online and real-time learning control system [101]. Reinforcement learning is an algorithm where machines learn to utilize experiences gained through varying the parameters and improve the desirable behavior of the system by receiving rewards (by feedback technique). This algorithm has been used in various applications to achieve human-level control, such as enabling autonomous robot and vehicle control, playing computer games, etc. [20][23][28][101][102].

Reinforcement learning aims to find a suitable sequence of action by which maximum reward can be achieved, leading towards an optimum outcome. Thus, it should be the right approach to soft robot automation. Conversely, the main objective of supervised learning is to extrapolate or generalize the response through training and produce correct responses for new data. This feature is not applicable for a system that needs to learn from its interaction with the dynamic environment – after changing the environment, an agent still shows adaptive performance, if it learns from its experience. On the other hand, unsupervised learning only discovers correlation within data, which cannot be a tool to maximize the reward signal, the key to generate self-learn performance [20].

Deep learning, a subcategory of machine learning, was introduced in 2006 [104]. In comparison with contemporary machine learning, deep learning eliminates the requirement of manual feature extraction; instead, it generates these features automatically. "*Deep learning is about learning multiple levels of representation and abstraction that help to*

make sense of data such as images, sound, and text”, said by R. Buyya *et al.* [105]. In the next sections, the thesis will discuss about deep learning and its coordination of reinforcement learning.

3.3 Reinforcement learning

Reinforcement learning is a process in which an agent learns to adjust to actions for receiving maximum rewards. The agent has no predefined knowledge of choosing an action, but it needs to figure out the suitable action which can bring out the highest reward by trying [20]. Reinforcement learning works in a dynamic environment and tries to find out the most suitable sequence of actions. In many cases, actions may affect the current rewards and, eventually, all subsequent rewards, leading to a “delayed reward”. Interactions with the environment through “trial and error” and “delayed reward” are two distinct features of RL [20].

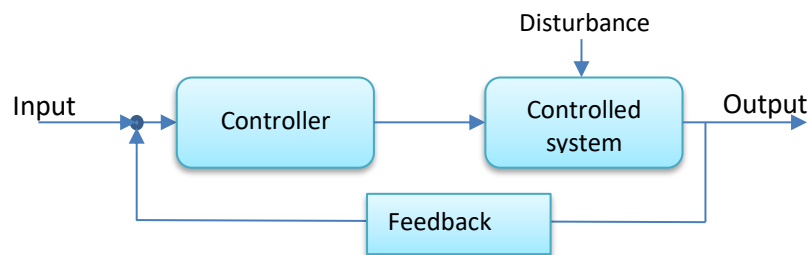


Figure 8. *Classical control system*

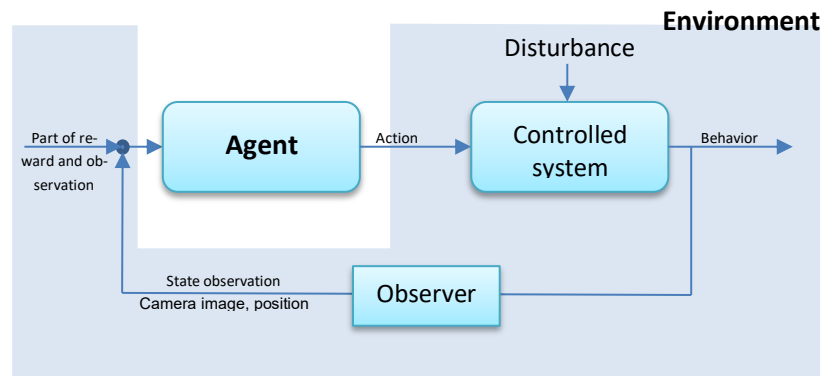


Figure 9. *RL in control system perspective*

In Figures 8 and 9, RL is compared to the classical control system. In a closed-loop classical control system, the controller gets feedback to correct the random disturbance and errors, thus improving and stabilizing the system. Whereas, RL system gets state observations and rewards as feedback because of the action it has taken and updates the agent according to the feedback to achieve maximum rewards. Sometimes the design of a traditional control system may become tough due to the nonlinearity or large

state and action space. In this kind of situation, RL is the best alternative. Because traditional control systems and RL have one common target to generate desirable system response [106].

3.3.1 Elements of reinforcement learning

RL environment refers to every component in the system except the agent. The agent is a piece of software that is responsible for generating action commands and sends it as an input to the environment, updating the policy through receiving observations and rewards from the environment. RL framework (Figure 10) has four basic elements connecting the agent and the environment. They are a policy, a reward signal, a value function, and a model of an environment (optional part) [20][106].

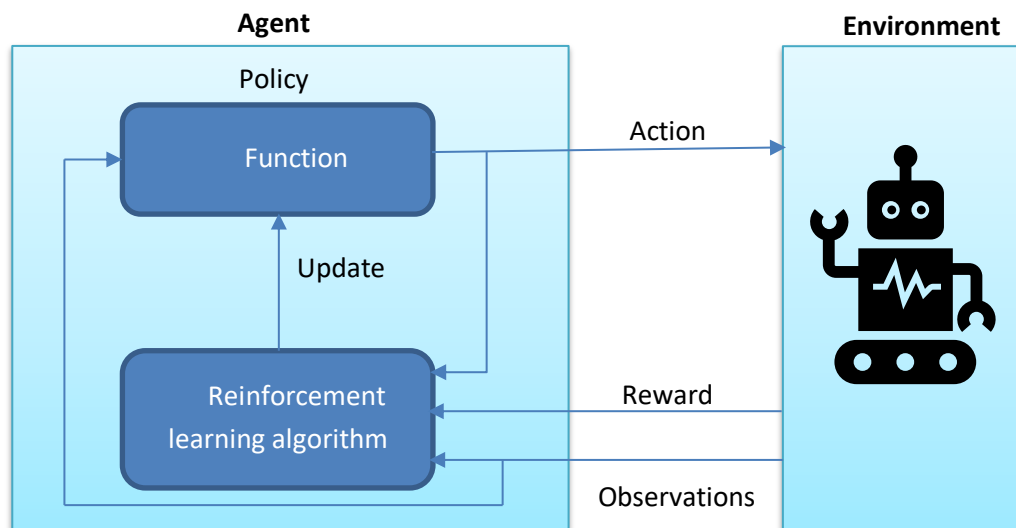


Figure 10. *RL framework and sub-elements*

Policy: Agent has a function that receives observational states, maps these states with actions, and decides the suitable actions to be taken. This function is referred as a policy. Policy is the core element of RL and solely responsible for determining the behavior of the system. A policy can be a lookup table of a simple function or a function incorporating extensive and complex computations. For generating optimal policy, RL algorithm is used. The RL algorithm can alter the policy depending on the actions, observational states and the amount of reward [20][106].

- **Reward:** On every time step, an action may change the observational states, after which the environment assigns a number by evaluating the performance quality or goodness of the behavior. This number is called reward signal. A reward function defines the aim of an RL problem, which is the central bias for

changing the policy. An agent has a crucial intention to collect the maximum total reward points over the long run. If a particular action chosen by the policy produces a lower reward, the policy will be altered by the RL algorithm in a way that it will take another action when a similar situation is met [20][106].

- **Value function:** Value function estimates the total future reward available for a sequence of actions from the current state and can bring the largest number of reward points in the long run. Actions are chosen depending on value judgment. Reward is related to the immediate result, whereas value function is related to the long-term results. For example, A particular state may produce a smaller immediate reward but can have high value because the state is a part of a sequence of actions that produce largest amount of reward and vice versa [20][106].
- **Model of an environment:** Model is an optional element and helps in planning, i.e., it serves a way of deciding the future effect of taking action. RL problems which use model and planning is known as model-based methods. On the other hand, RL problems which use trial and error methods instead of model or planning is known as model-free method [18][20][106].

By using RL algorithm, the agent eventually learns the best policy, which is able to take optimal action sequences to generate maximum rewards [20][102]. RL tasks can be continuous or episodic. In continuous tasks, the interaction between agent and environment does not break down. Conversely in episodic tasks, the interaction breaks down into several separate episodes where episodes end after finite time steps regardless of achieving the goal or not. This one is mathematically efficient because the effect of a particular action on subsequent finite reward can be determined in every episode [18][20].

3.3.2 Markov decision process (MDP)

Robotic function, which is a sequential task through interaction with the environment to achieve an objective, can be represented as Markov Decision Process (MDP). MDP includes a tuple $\{A, S, R, P\}$; where A is a set of actions, S is a set of observational states, P is a state transition probability function, and R is a reward function. Sometimes, a discount factor, γ , is used [18][20]. In RL, the interaction between agent and environment happens in discrete time steps, $t = 0, 1, 2, \dots$. At any t , agent receives observational states, $s_t \in S$, and sends an action, $a_t \in A$. At $t+1$, because of the a_t , agent gets a reward, $r_{t+1} \in R$ as well as a new state s_{t+1} . At every step, policy π_t is updated. At a time step t , $\pi_t(s, a)$ is the probability of $a_t = a$ when $s_t = s$. Also the reward function and state transition function

can be denoted as $R(s,a)$ and $P(s_{t+1}, a_{t+1})$ accordingly. In an episodic task, one episode of MDP can be represented as $\{(s_0, a_0, r_1), (s_1, a_1, r_2), (s_2, a_2, r_3), \dots, (s_{t-1}, a_{t-1}, r_t)\}$ where the episode terminates with terminal state s_t . According to MDP, the probability of any state s_{t+1} depends only on the previous state s_t and action a_t [22][107][108]. A typical requirement for MDP is that the robot should fully understand the entire observational states, which is in practice difficult. However, RL can deal with model-free and partially observable MDP [20][108].

3.3.3 Reinforcement learning algorithms

RL framework has mainly two classes of algorithms for solving problems, which are value-based methods and policy-based methods. There is also a third kind of hybrid approach, Actor critic methods. These techniques can also extend with deep learning [109].

- **Value-based methods** estimate the expected return value of taking an action in a given state. For instance, Bellman's equations are used to estimate the sequential states. These methods incorporate with Q-learning and SARSA (State-Action-Reward-State-Action), though they may differ in target values. In Q-learning, the target value, *i.e.* Q-values are recursively updated at each time step. Q-learning is an off-policy method. On the contrary, SARSA updates the value estimation using a policy and is an on-policy method[18][109].
- **Policy-based methods** do not use value estimation, but policy can be updated directly through evaluation and improvement. These methods can be gradient-based or gradient-free depending on parameter estimation. They have some advantages over value-based methods, such as convergence, less computational time, dealing with continuous high-dimensional data, and solving deterministic policies effectively. However, these methods are not suitable for a dynamic environment where the agent needs to adapt [18].
- **Actor critic methods** can carry out a distinct representation of policy and state estimation, and combine the iterative techniques of value function and policy-based methods [18]. The actor, *i.e.* policy, learns by getting feedback from the critic, *i.e.*, the value function. Mainly, these methods utilize the value function as a baseline for policy gradients. Hence, an actor-critic method uses learned value function which is the main difference comparing with other two methods [109][110].

3.3.4 Value function

Agent aims to maximize future rewards by selecting suitable actions wherein a discounted factor γ is applied to rewards [20][111]. The future discounted return at time t is

$$R_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$$

where r_t is the reward at time step t received after taking an action and T is the terminal time step of the episode. After some sequence of action, a , and state, s the optimal action-value function is defined as

$$Q^*(s, a) = \max_{\pi} E [R_t | s_t=s, a_t=a, \pi]$$

where π is the policy. Here $Q^*(s, a)$ obeys the Bellman equation, an essential property of dynamic programming. If the optimal value of the sequence s' at the next step is known for all possible actions a' , then the optimal strategy is to choose the action from all the possible actions which will maximize the expected value of $r + \gamma Q^*(s', a')$,

$$Q^*(s, a) = E [r + \gamma \max_{a'} Q^*(s', a'), | s, a]$$

The action-value function is the foundation of many RL algorithms. The function utilizes Bellman equation for iterative updating of

$$Q_{i+1}(s, a) = E [r + \gamma \max_{a'} Q_i(s', a'), | s, a]$$

And when $i \rightarrow \infty$ then $Q_i \rightarrow Q^*$; i.e., the value function iteration converge to the optimal action-value function [20][111]. For every sequence, a separate action-value function is estimated, which is not practical for applications. So a typical approach is to utilize a function approximator for general estimation of action-value function [20][22][103][111]. Linear function approximator and nonlinear function approximator such as neural network can be used in RL, though the nonlinear function approximator has been most commonly used [22]. In the next sections, Q-learning is introduced, followed with incorporation with deep neural networks for developing more efficient learning method.

3.3.5 Q-learning

The core in reinforcement learning is temporal difference (TD) [112] and learning action-value function from direct experience with TD error using the following update rule[107][111]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Where α refer to learning rate. Here, the learned action-value function, Q directly approximate the optimal action-value function, Q^* . So it is not dependent on any policy *i.e.* off-policy control, which enables early convergence [20][112].

3.4 Deep reinforcement learning

Usually, a robotic system has a large number of DOF, large dimension of continuous observational states and action space, and sometimes accompanied with high noise. A similar challenge was faced by Mnih *et al.*, in which they tried to play a computer game called Atari through reinforcement learning [22]. The game has 10^{67970} possible game states, which is a significantly large number [111]. The challenge was to scale up the dimension of action and observational space as traditional RL based on MDP [113]. For solving this kind of challenge, deep learning plays an important role where the neural network is used to extract features from highly structured data.

Reinforcement learning can use deep neural networks for approximating different components, such as value function $V(s, \theta)$ or $q(s, a; \theta)$. Policy $\pi(a|s; \theta)$ and model, *i.e.*, state transition and reward. Here θ is the weight parameter of the neural network [111]. However, RL with neural network used to approximate the action-value function is not very stable. Significant changes occur in the policy due to the small shift in action-value function (Q), and this leads to changes in the data distributions and the correlations of Q value and also target value [114]. For tackling these problems, the most common strategy is experience replay where all experiences are stored in a replay memory. During training, random mini-batches of replay memory replace the most recent transition which removes the correlation in the observation sequence and changes the subsequent training sample to avoid the local minimum [111].

In RL, the agent needs to find a policy through trial and error method to fetch more rewards, which raise the idea of exploration and exploitation. Exploration is related to try new strategies, *i.e.* try a few new choices on top of existing information to explore further information about the environment; Exploitation is associated with maximizing the reward using known information. Meanwhile, typically epsilon-greedy strategy is used to ensure feasibility between both [103][107].

3.4.1 Deep Q-learning

A Q-network can be designed using neural networks with weights θ . For applying experience replay, experiences of the agent at each time step, $e_t=(s_t, a_t, r_t, s_{t+1})$ is stored into a

data set $D_t = \{e_1, \dots, e_N\}$ [22][103][107]. The deep Q-learning update iteration, i , using following loss function,

$$L_i(\theta_i) = E_{(s,a,r,s')} [(y_i - Q(s, a; \theta_i))^2]$$

Where $y_i = r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$ and weights, θ_i^- are used to calculate the target at i . To estimate the action-value function without any generalization,

$$Q(s, a; \theta) \approx Q^*(s, a)$$

The gradient of the loss function (differentiating with respect to θ_i) is,

$$\nabla_{\theta_i} L(\theta_i) = E_{s,a,r,s'} [(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i)]$$

Though the full expectation can be computed from the above equation, optimization of the loss function using stochastic gradient descent is more convenient. Therefore, a similar approach as Q-learning can be used for DQN framework, where the weight is updated after each time step, and the expectation is replaced by a single sample by setting $\theta_i^- = \theta_{i-1}$ [103][107]. The deep Q-learning algorithm with experience replay is shown in Figure 11.

```

Initialize replay memory  $D$  to capacity  $N$ 
Initialize action-value function  $Q$  with random weights  $\theta$ 
Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$ 
For episode = 1,  $M$  do
  Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$ 
  For  $t = 1, T$  do
    With probability  $\epsilon$  select a random action  $a_t$ 
    otherwise select  $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$ 
    Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$ 
    Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ 
    Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $D$ 
    Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$ 
    Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$ 
    Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  with respect to the
    network parameters  $\theta$ 
    Every  $C$  steps reset  $\hat{Q} = Q$ 
  End For
End For

```

Figure 11. *The deep Q-learning algorithm with experience replay. Adapted with permission from Ref. [103]*

DRL approach combines RL and deep learning to tackle different challenges of traditional RL approach like dependency on handcrafted features, discretized input and output spaces, and the lack of scaling up the dimensionality of input and output states

[25][113]. Decision-making performance is enhanced by incorporating deep neural networks. In 2013, DeepMind Team of Google successfully utilized Q-learning with deep neural networks *i.e.*, deep Q learning to tackle the low dimensionality of input and output states problem [22]. They utilized deep Q learning to play computer games like Atari [22]. Following the milestone of the DeepMind team, many successful experiments have been conducted by developing an optimized control policy in an uncertain environment, which is suitable for autonomous systems [23]. For instance, mobile robot path planning and autonomous navigation [23]–[26], autonomous driving [19][27], robot motion control[28][29], just to list a few. Deep Q learning approach can be applied in controlling locomotion and navigation for different soft robotic systems that integrate the path finding capability as well as their mechanical properties [18]. In deep Q learning approach, the controller directly learns from the raw data while interacting them in a dynamic environment and provides an end-to-end solution [11][13]. As deep Q learning approach does not consider structure and material of the robot [3], a light responsive LCN robot can be considered as a good platform to test deep Q learning. For autonomous control of locomotion in a dynamic environment, a reward system and optimal control policy are developed, as will be discussed in the next chapters of the thesis.

4. IMPLEMENTATION

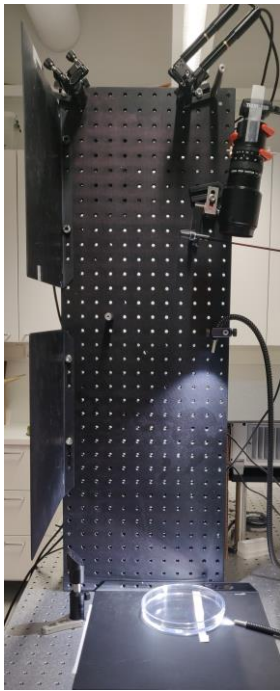
As illustrated before, LCN robots can be driven by light, and each part of the robot can be actuated/controlled upon a laser or an LED light field, for creating robotic movement. For precise control, in general a laser beam is suitable for pin-pointing controlling of robotic segments, such as robot's walking legs or deforming body. For utilizing this photo-actuation feature, a workspace is set up to steer the laser beam within two-dimensional plane. This workspace is beneficial for conducting different kinds of experiments to explore the different robotic capabilities. In this chapter, the laser steering workspace setup and its components will be firstly presented, followed with robot tracking system.

4.1 Laser steering workspace setup

The laser steering workspace configuration is divided into two parts: machine vision, and laser steering subsystem. The workspace setup and its different components are shown in Figure 12.

4.1.1 Machine vision subsystem

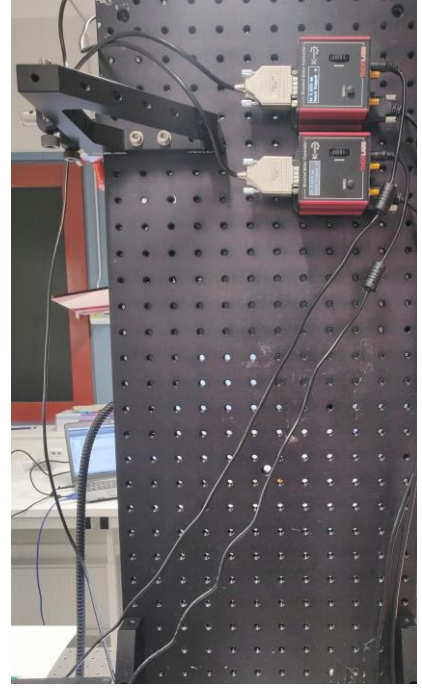
In this workspace, a Zeiss Cold light source (CL 4500 LED CRI90) is used as an illumination source that can serve up to 450 lm light flux with a continuous dimming option [115]. A monochrome camera sensor (DCC1545M, Thorlabs [116]) equipped with a zoom lens (Zoom 7000, Navitar [117]) is used to capture images. This camera has a CMOS sensor of 1.3 Megapixels resolution with an electronic rolling shutter (see technical details in Table 2). The zoom lens is a close-focusing macro lens with a minimum working distance of 13 cm, and can be manually controlled to adjust the zoom-in factor and tune the focus. The objective of the camera is to capture live images of the working platform while the laser beam is steering on the robot. However, the brightness of the laser beam hinders the structural information of the robot (Figure 13b). In order to solve this problem, a colored glass filter is used in front of the lens to block the laser wavelength (532 nm), yielding a good-quality image suitable for further data analysis (Figure 13c).



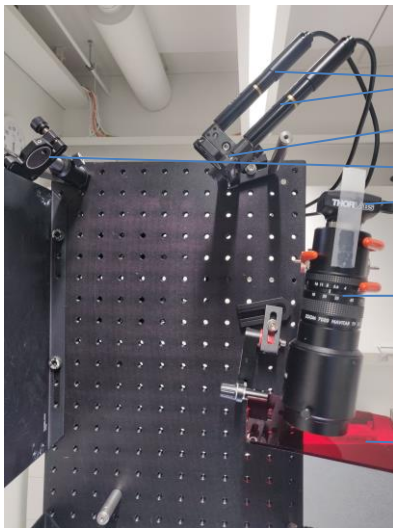
Front view



Side view

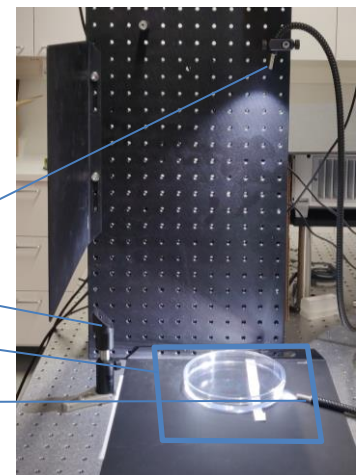


Rear view



Front closer view

- Servo motors
- Adjustable mirror
- Mirror
- Camera sensor
- Lens
- Optical Filter



Front closer view

- Illumination
- Mirror
- Workspace
- Illumination



Rear closer view

- Servo motor controllers

Figure 12. Laser steering workspace setup (front, side and rear view) and different components of the workspace

Table 2. Camera specifications[116]

| Parameter | Value |
|-----------------------|------------------------------|
| Model | DCC1545M |
| Sensor | CMOS, Monochrome |
| Resolution | 1.3 Megapixels (1280 x 1024) |
| Exposure Mode | Rolling Shutter |
| Optical Sensor Format | 1/2" |
| Read Out Mode | Progressive Scan |
| Frame Rate | 25 fps |
| Dynamic range | 68.2 dB |
| SNR(MAX) | 45 dB |
| Trigger | Input: No, Output: Yes |
| Lens Mounting Thread | CS-Mount |
| Interface | USB 2.0 |
| Supply voltage | 3.0 V–3.6 V, 3.3 V nominal |
| Dimensions | 48.6 mm x 44 mm x 25.7 mm |
| Weight | 32 g |

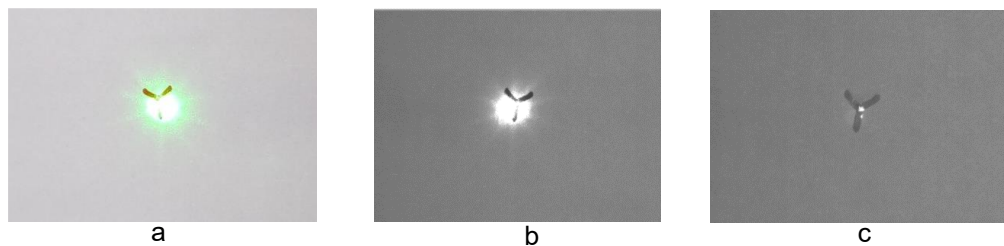


Figure 13. *a. An LCN robot and the robot's leg illuminated with a green laser beam (captured using a regular camera). b. Image from the machine vision system without an optical filter. c. Image from the vision system's camera with a colored glass filter*

4.1.2 Laser steering subsystem

For the actuation of an LCN robot, a solid-state, continuous-wave green laser (532 nm) is used, and the power of the laser can be controlled manually, see technical specifications in Table 3. The laser beam is guided to an adjustable mirror mounted on top of the workstation by using several mirrors. The adjusted mirror (SM05, Thorlabs[118]) is driven by two DC servo motors and can continuously change/monitor its angular displacement via computer control. The servo motor actuators can travel up to 12 mm, and the step resolution is in a submicron level which allows a continuous angular displacement of the mirror, see details in Table 4. Eventually, the laser beam is reflected down to the workspace, and steered on the 2D plane along vertical and the horizontal direction, allowing precise robotic actuation (Figure 12).

Table 3. Solid-state green laser [119]

| Parameter | Value |
|--------------------------|-----------|
| Model | MGL-F-532 |
| Wavelength (nm) | 532±1 |
| Output power (mW) | 2500 |
| Transverse mode | TEM00 |
| Operating mode | CW |
| Power supply (90-264VAC) | PSU-H-LED |

Table 4. DC servo motor actuators (Z812 from Thorlabs)[120]

| Parameter | Value |
|-----------------------------|---------------------------|
| Model | Z812 |
| Travel Range | 12.0 mm |
| Motor Type | 6 VDC Servo |
| Micro steps per Revolution | 34304 |
| Backlash | <8 µm |
| Bidirectional Repeatability | <1.5 µm |
| Home Location Accuracy | <2 µm |
| Velocity | 2.6 mm/s (Max) |
| Acceleration | 4 mm/s ² (Max) |
| Weight | 0.134 kg |

Note that the controller of the servo motor provides a differential encoder feedback to ensure accurate positioning operation. Compatibility with the ActiveX® programming environment makes this controller suitable for developing custom applications. The details of the servo motor controller can be found in Table 5.

Table 5. Servo motor controller (KDC101 from Thorlabs) [121]

| Parameter | Value |
|------------------------|--|
| Model | KDC101 |
| Drive Voltage | ±12 to ±15 V |
| Drive Type | 8-bit Sign/Magnitude PWM |
| Control Algorithm | Digital PID Filter |
| Feedback | Differential Encoder Feedback (QEP Inputs) for Closed-Loop Positioning |
| Velocity Profile | Trapezoidal |
| Software Control | Kinesis® or APT™ |
| Output | 15 V/ 2.5 W |
| Interface | USB 3.0 |
| Dimensions (H x W x D) | 60.0 mm x 60.0 mm x 49.2 mm |

4.2 Robot tracking system

The robot tracking is a synergistic system between machine vision and servo motor control subsystem. The integration is achieved through a communication server, as shown in the schematic drawing in Figure 14. The machine vision is responsible for spotting the robot location as well as extracting feature information of the robots, such as the location of the legs, body orientation, the destination location, etc. The servo motor control subsystem receives information from the machine vision and automatically steers the laser beam to the desired location according to some pre-designed algorithm. The following sections will describe all these systems in detail.

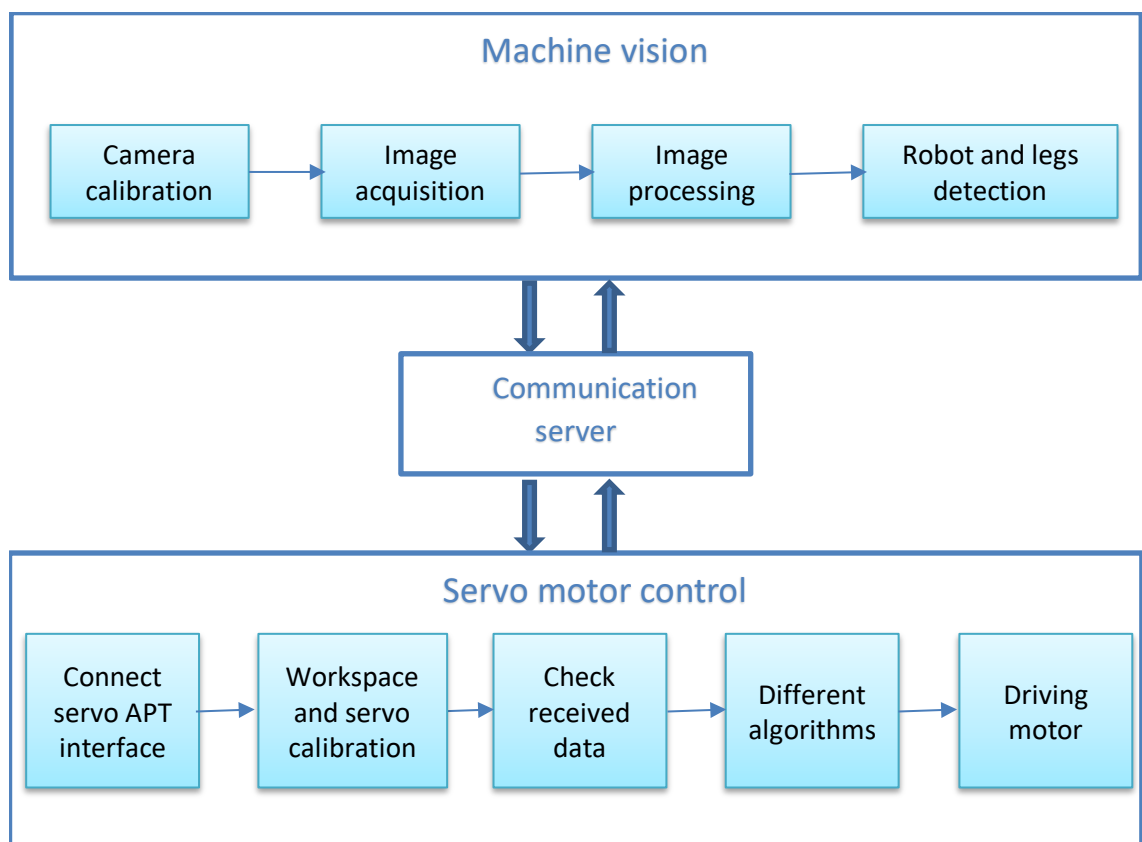


Figure 14. Robot tracking system

4.2.1 Machine vision for robot tracking

For tracking the robot position, a dimensional calibration step is taken before using the vision function in robotic applications [122]. After that, all images captured by the camera contain dimensional information in real space. The captured images are further processed with digital image processing and feature extraction technique, in order to enhance different structural features. For this, NI Vision Builder for Automated Inspection (Vision Builder AI), an application from National Instruments is used in calibration including programming and configuring vision algorithms for image analysis and processing.

Camera calibration is essential for precise detection and high accuracy in measurement. It establishes a relation between the image and the actual scene for real-world measurements. For instance, distortion due to perspective errors and lens aberrations affects image coordinate and thus the related geometrical measurements [123][124]. Camera calibration helps to correct these errors and to provide accurate dimensional data.

Vision Builder AI application is used for camera calibration by utilizing a dot grid pattern. The grid has circular dots with equal spacing in both horizontal(x-axis) and vertical(y-axis) directions. By default, this application provides measurements in pixel units. Spatial calibration, *i.e.* mapping pixel into real-world units is also included in this calibration. A distortion model (grid) was used in the calibration process. The dot grid pattern was captured (Figure 15a) using the machine vision setup. The center to center distance between dots was provided including a user-defined reference coordinate to transform the pixel coordinate to real-world coordinate (Figure 15b). The center to center distance between dots was 9.3053 mm in both direction x and y.

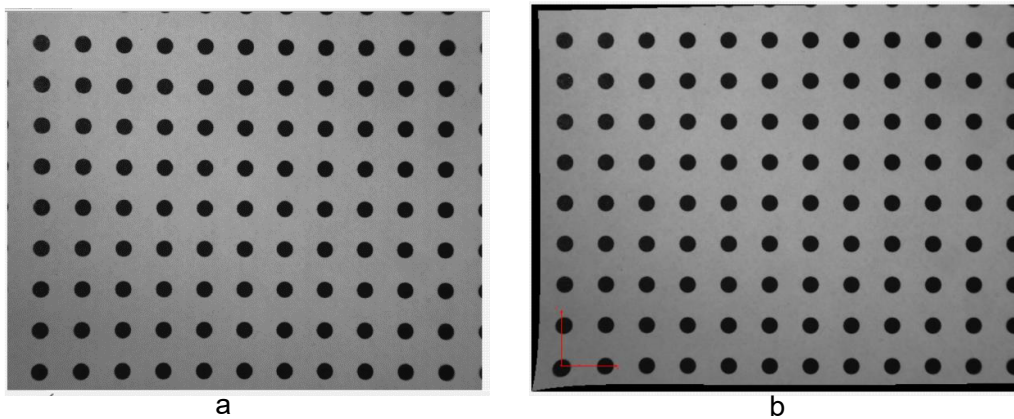
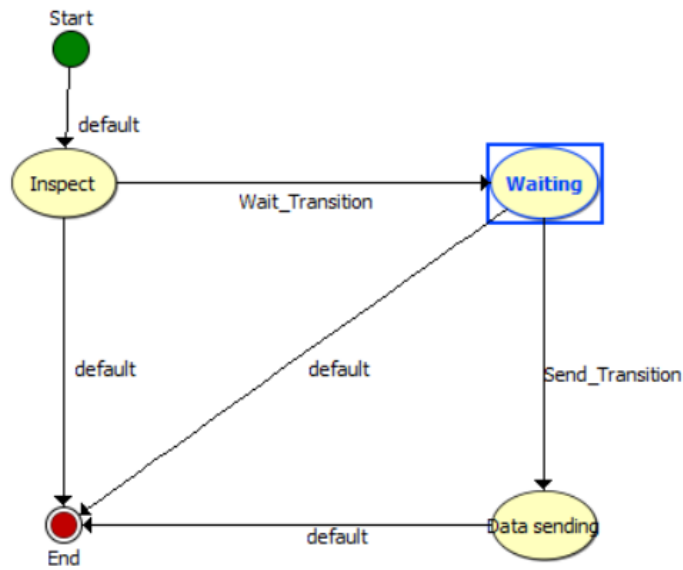


Figure 15. a. The dot grid pattern for camera calibration b. Image after calibration, including the tangential distortion correction

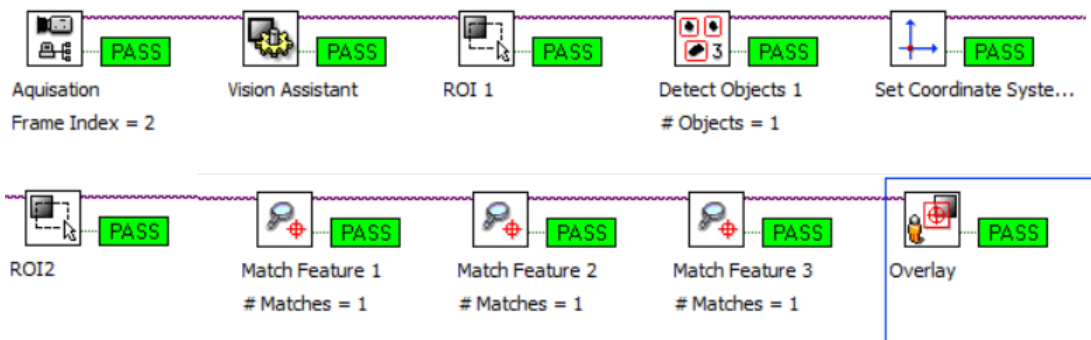
After this initial calibration process, the system is ready to capture inspection images for further image processing. Vision Builder AI uses a state diagram to model the image analysis and feature extraction process, and every state can have several steps and transitions. The developed state diagram is presented in Figure 16a.

The inspection starts with the *Start* state and immediately makes a transition into the *Inspect* state. The *Inspect* state has several steps (Figure 16b). The first step is *Acquisition*, in which an image is captured from the camera. The second step, *Vision Assistance*, processes the image to enhance the desired feature by using Color Threshold, Lookup Table, Smoothing and Diate technique (Figure 16c).

a



b



c

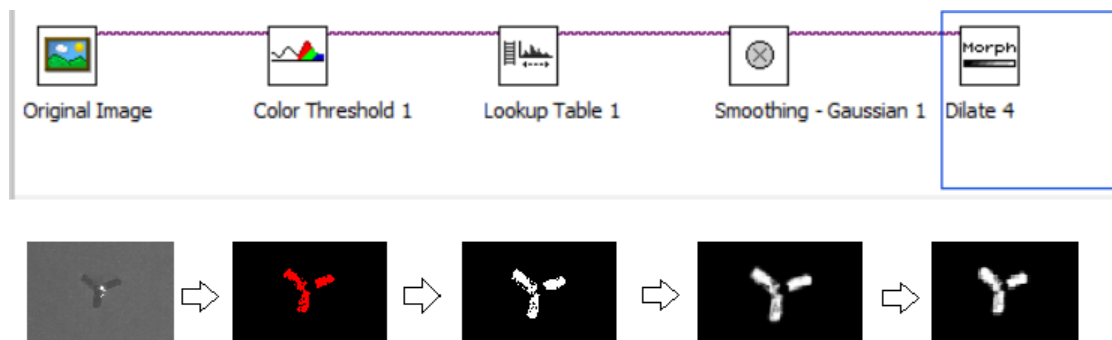


Figure 16. a. State diagram for the vision system. b. Steps in the Inspect state. c. Steps under the Vision Assistant step and changes of the original image along with these steps

After that, the robot is detected (*Detect Objects 1* step, Figure 16b) within a predefined region of interest (*ROI1* step) and then creates another region of interest (*ROI2* step) around the robot body (Figure 17). *Match feature 1*, *Match feature 2* and *Match feature 3* steps extract distinct and desired features from ROI2. The last step of the *Inspect* state is *Overlay*, which is responsible for presenting information about the detected features for the user (Figure 16b).

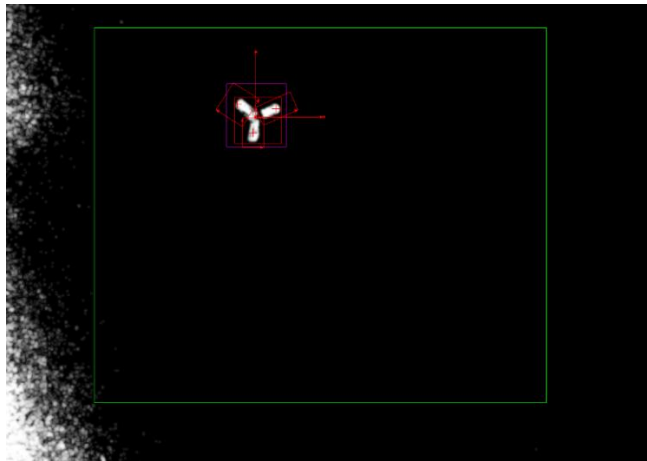


Figure 17. Output of the *Inspect* state and rectangular boxes indicate some steps (i.e. green: *ROI1*, purple: *ROI2* and red: *matched feature*)

After completing these steps, the *Inspect* state makes a transition depending upon the 'PASS' value of all previous steps. If every step of the *Inspect* state is executed successfully (pass), then the current state will take the *Wait_Transition* to the *Waiting* state; otherwise, the inspection will end by taking the *default* transition (Figure 16a).

In the *Waiting* state, a communication block waits for a fixed period to receive a specific string from the communication server. Within this time window, if the communication block has not received the specific string, the *Waiting* state transits to the *End* state. Otherwise, it transits to the *Data sending* state where another communication block sends the information about extracted features and parameters to the communication server. Extracted features and parameters can be the locations of the matched features, i.e. that x and y coordinate values of robot legs and target, scores of matched patterns (not for every case), also a check string to notify the ending of the message. After sending the message, the *Data sending* state transits to the *End* state. When the execution reaches the *End* state, the inspection starts again from the *Start* state.

Figure 16 represents a basic state diagram used in this thesis. For implementing several case studies with different types of robots, some modifications are adopted in the state diagram, especially in the *Inspect* state and the *Data sending* state (inspection programs are available at this link: https://github.com/amankhan47/Aman_SPM_TUNI.git).

4.2.2 Communication server

The machine vision and servo motor control subsystems are connected through a TCP IP communication server. After feature extraction, the developed inspection program of the vision system waits for a fixed period. Within this period, upon receiving a string from the servo motor control subsystem, a message is sent to the servo motor control subsystem. The servo motor control subsystem does not assign a sending string until the previous operation of laser steering being finished.

4.2.3 Servo motor control subsystem

The servo motor control subsystem essentially deals with the control of two servo motors, to steer the reflected laser spot to the designed positioning according to the messages from the communication server. Particularly, one servo motor creates steering movement along the x-axis, and the other along the y-axis. Hence, the motor-driven spot motion requires another calibration process to connect the distance information (from machine vision) with the laser position (driven by DV servo motor current) in the workspace. The servo motor control subsystem is designed as an object utilizing MATLAB application. The object contains methods for every component of this subsystem (Figure 18 and Appendix A).

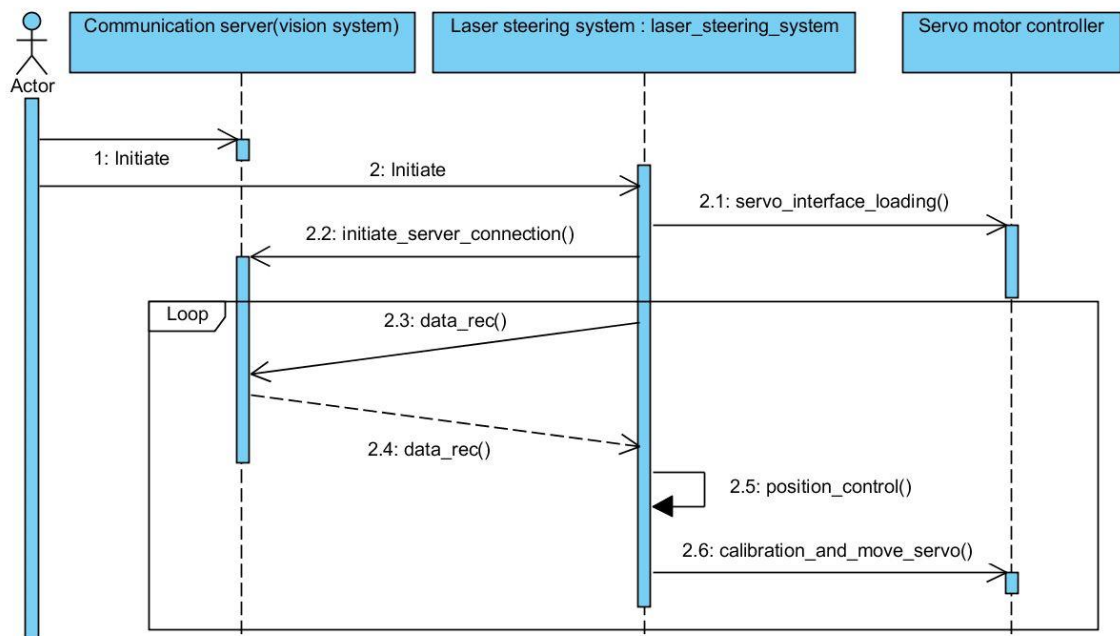


Figure 18. A sequence diagram of the servo motor control subsystem

The ActiveX® environment compatibility makes the servo motor controller suitable for developing applications in MATLAB. The *servo_interface_loading()* method establish connection to the controllers through ATP server by loading a servo control interface,

sending initiating commands, setting the serial numbers of the controllers, and finally moving the motors to home positions. After this method, the *initiate_server_connection()* method establishes a TCP IP connection with the communication server. The *data_rec()* method requests for the extracted feature message by sending a string to the communication server. After receiving the message, the *position_control()* method is responsible for selecting suitable leg locations and pass the information to the *calibration_and_move_servo()* method. This method starts steering the laser beam to the desired location. In this method, calibration requires some dimensional parameters of workspace and servo motors. They are the moving distance of the steered spot in both x and y directions (x_{in_mm} , y_{in_mm}) and the motor positions at starting and finishing points (x_0, x_f, y_0, y_f) of that trajectory. For reaching a specific location (x_{move} , y_{move}) monitored by the vision system, servo motors need to reach the following positions,

$$x_axis = (x_0 - ((x_0 - x_f) * x_{move}) / x_{in_mm}) \text{ and}$$

$$y_axis = (y_0 - ((y_0 - y_f) * y_{move}) / y_{in_mm}),$$

here, x_axis and y_axis are the motor positions of x-axis and y-axis motors respectively. Using these values, the laser beam is steered to a specific location. After reaching the location, this program again starts executing from *data_rec()* method to *calibration_and_move_servo()* method in a loop that ensures continuous operation in the experiment (Figure 18).

An optional method, *set_k()* is also developed, which allows manual selection of the leg locations and assistance in evaluating the locomotion potential of an LCN robot before developing a control strategy for continuous operation. Besides, the *position_control()* method can be altered to develop other control strategies, depending on different geometries in the robot design.

The machine vision subsystem assists in deploying different vision algorithms to extract message about robot's structure, which contains shape or configuration information. The servo motor control subsystem utilizes this information to steer the beam and actuate the robot, depending on the adopted locomotion control strategies. We believe the robot tracking system has the potential to be utilized in many micro-robotic applications. Furthermore, this system is capable of being integrated with reinforcement learning for automatic locomotion control, which will be discussed later.

5. EXPERIMENTAL RESULTS

Different soft robotic structures can be realized by using photomechanical LCN actuators to achieve versatile robotic functions, such as shape-change, walking, and swimming. Basically, these robots are often constructed by cutting actuator strips from an LCN film and pasting them into designed configurations. The movement is then created by the out-of-plane deformation of a bending strip. In this chapter, I will first characterize the deformation properties of a splayed bending LCN strip. Then, I will introduce a failed trial of my study in investigating locomotion of an LCN walking device, and a successful one in realizing swimming locomotion on the water surface.

5.1 Light actuation in an LCN bending strip

In most cases, LCN strip is the basic building block of LCN robots, providing reversible shape-change by bending actuation. Here, an experiment was conducted to examine the bending deformation of an LCN strip. An LCN strip was prepared with splayed alignment [53], and attached to a base (Figure 19). The whole strip was irradiated with a white-light illumination source (from a projector with lenses, about 1 W/cm^2). A video analysis tool was developed and used to measure the deformation angle and capture the frames of the bending deformation (Appendix B). The measured angle is indicated in Figure 19b and c. Figure 19a shows the images of deformed strip when light is switched on and when ceasing the light, while Figure 20 plots the measured deformation angles, automatically collected by the analysis program.

The LCN strip is heated up upon irradiation, and the deformation is driven by anisotropic thermal expansion between the two surfaces of the splayed film. At 1.4 s (Figures 19a and 20) the strip was irradiated with light, and immediately it started to bend. At 3.2 s, the strip touched the substrate, which blocked further deformation. In this situation, the portions close to the base and the strip-tip were fixed, bringing out no change in the measured angle, however, other sections of the strip still deformed until 5.0 s, where the light-induced heating was saturated. In total, the LCN strip bent 242° within 1.7 s. After ceasing the light, the strip relaxed back to its original shape after about 30 s.

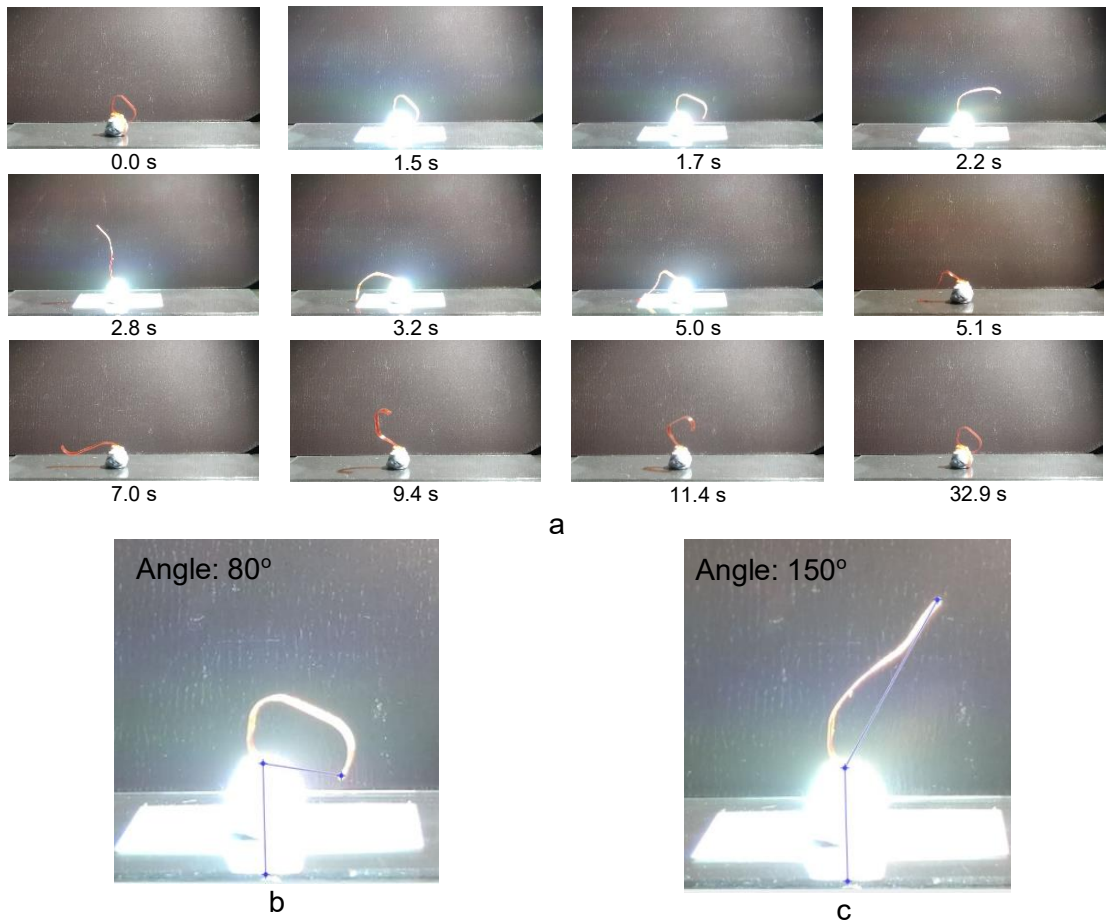


Figure 19. a. Timeline of LCN strip deformation dynamics during (light on at 1.5 s) and after (light off at 5.1 s) light exposure. b, c. Angle measurement, the vertical blue line is the fixed axis and the other axis follow the tip of the strip

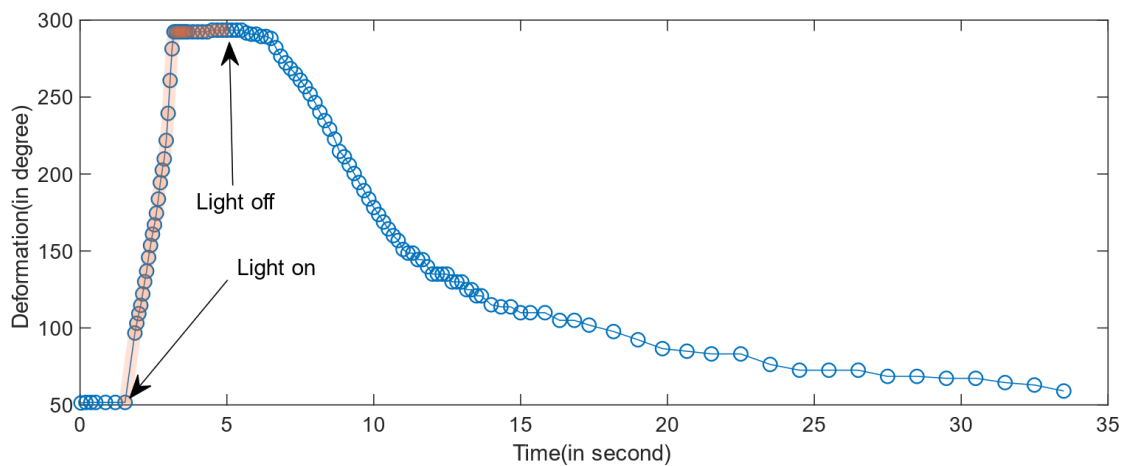


Figure 20. LCN strip bending deformation

The average angular velocity of bending was $137^\circ/\text{s}$ upon illumination (actuation speed) and $8^\circ/\text{s}$ after ceasing the light (relaxation speed). This experiment presents the impact of light on/off operation on an LCN strip, which is advantageous for devising soft robots, as illustrated in the following sections.

5.2 Optical control in three legs walking robot

The robot was built utilizing three pieces of LCN strips (2.5 mm x 1 mm x 50 micron) being UV glued into a central symmetric configuration (see the photograph in the insert of Figure 21). We have implemented machine vision and tracking system, as discussed in Chapter 4. To obtain walking locomotion, the robot was placed on a paper surface, while the machine vision subsystem could automatically detect the robot's legs, central position, and orientation (Figure 21). Particularly, the machine vision could also capture images from the top of the robot, measuring the length change of the legs. The decrease of leg length upon illumination indicates bending of the strip towards the ground, which is the key indication of leg actuation. In this experiment, the template matching technique was utilized for the detection of legs. The image processing tool was provided with the straight leg templates of the robot. Depending on the matches with the template in the inspections, image processing tools provided score values that were used as a measure of bending. All this information of detected features was sent to the servo motor control subsystem.

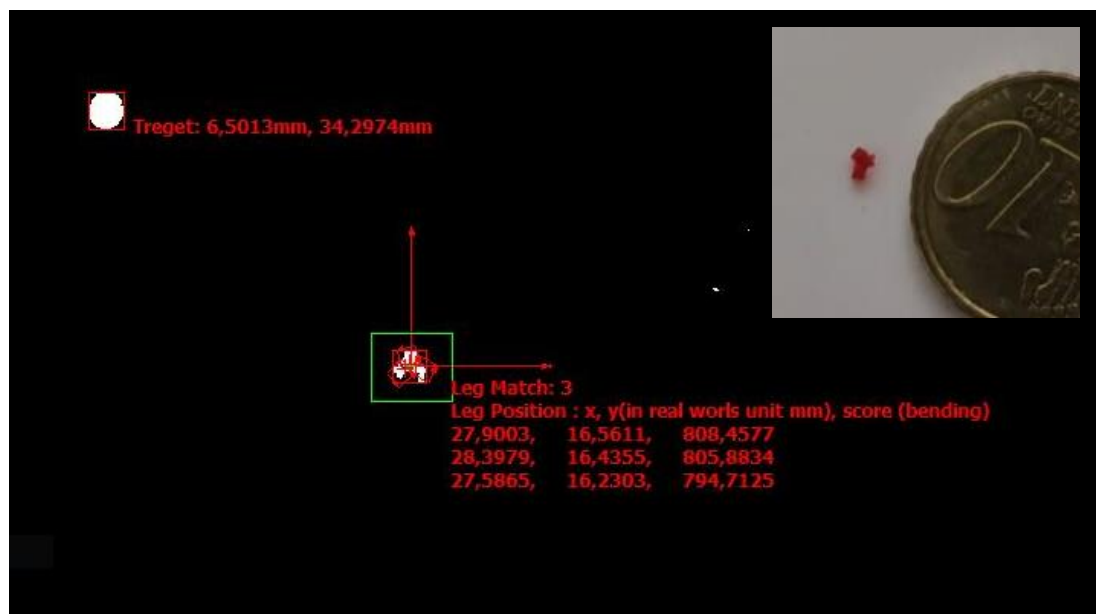


Figure 21. Features detection of three legs walking robot's through machine vision, photograph of the robot in the insert

Then, the servo motor control subsystem started processing the received information. For steering the laser beam, the `set_k()` method of servo motor control subsystem selected one specific leg to be irradiated by calling the method with 1, 2 or 3 as an input parameter (Appendix A). The subsystem re-tracked the robot position/orientation after finishing one steering step and continued with next steering operation. After several operations, the robot was expected to produce a significant translation through a sequence

of leg bending. However, unexpected hurdles have raised up during the experiments, such as strong adhesion between the soft material and the ground interface, and unpredictable randomness of friction. The robot has been tested on several substrates including paper, plastic, and glass; however, no controlled locomotion was achieved. Since dealing with kinetics of soft material is not the main objective of this thesis, in order to complete the study of machine learning, an alternative plan was chosen based on another type of locomotion, in order to produce more reliable light-controlled function.

5.3 Light propelled robotic swimmer

LCN strip can be used to build swimmer robots. A floating square film (2.5 mm x2.5 mm) was placed on the water surface for exploring the capability of light-propelled swimming. The mechanism is based on the fact that light illuminates on one or few edges of the film, thus heating up the structure asymmetrically. Raising the temperature decreases the force of surface tension on that edge, thus dragging the film towards the direction of the opposite edge. Eventually, the film is always floating away from the laser spot. To achieve automation in light steering of swimming towards the preselected destination, machine vision was used. More specifically, the machine vision subsystem was used to detect the location of two opposite edges of the square film as well as the destination information (a marked line). This information was sent to the servo motor control subsystem. The *data_rec()* method and the *position_control()* method of servo motor control subsystem were changed to match each specific situation. To deploy automatic control of laser steering, the *position_control()* method used following algorithm (Algorithm 1, code is available at this link: https://github.com/amankhan47/Aman_SPM_TUNI.git):

| Algorithm 1: |
|---|
| [1] Collect the edge location information |
| [2] Calculate the distance between the destination and the edge location |
| [3] Select the edge location which is farther away from the destination |
| [4] Call <i>calibration_and_move_servo()</i> method with selected edge location |

Using this algorithm, the laser beam always irradiated the edge of the film with longest distance to the destination (Figure 22). Meanwhile, machine vision subsystem detected the new location and steered the laser beam to the new location automatically. Eventually, the system was able to propel the film towards the destination line, with an average speed of around 0.66 mm/s and entire period of 15 s.

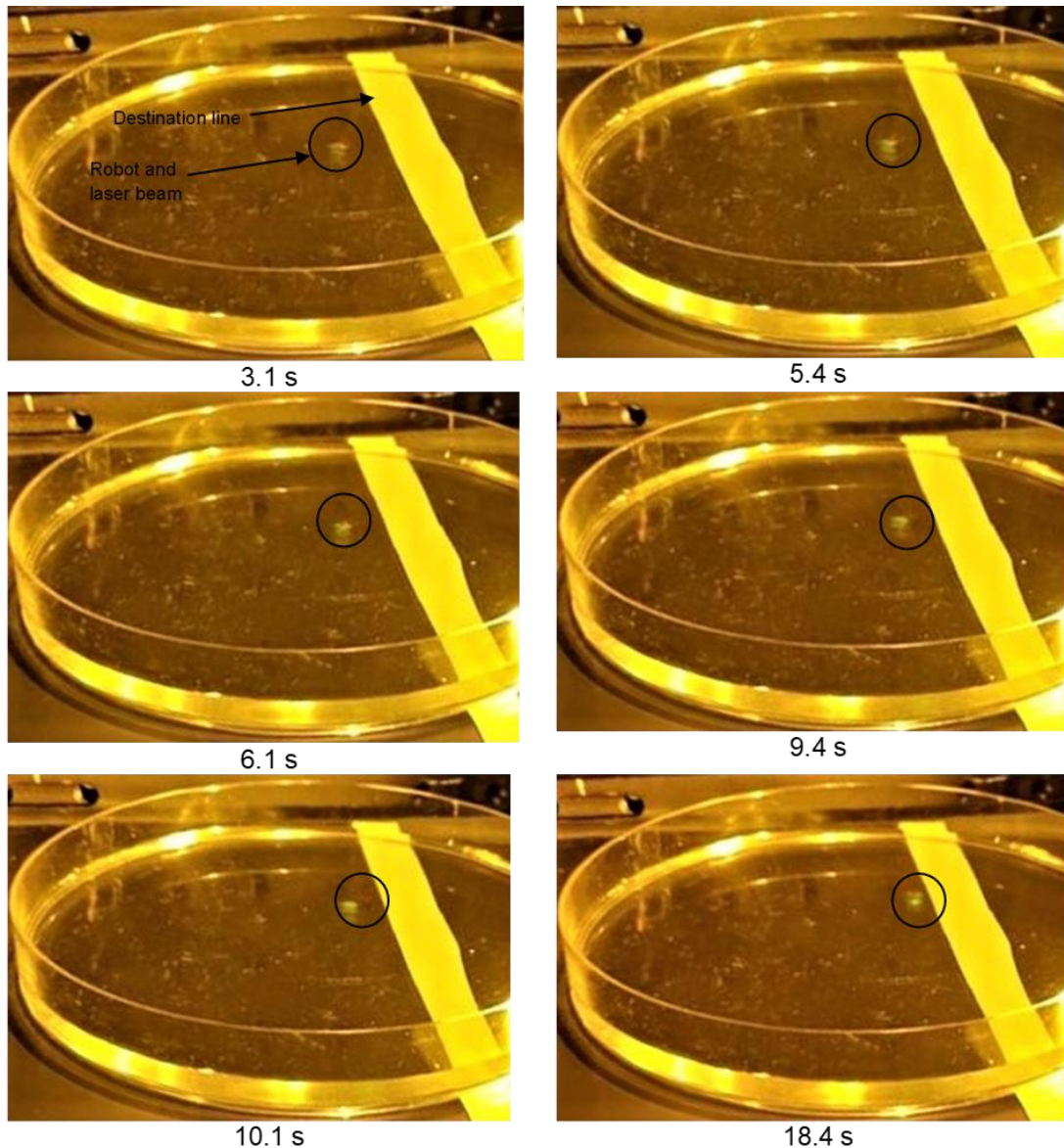


Figure 22. *Locomotion timeline of a light propelled robotic swimmer*

Though locomotion of the film encompasses some degree of random rotation, it reached to the destination line. The film can be propelled forward or backward by selecting edges of the film. This on-demand directional movement of this experiment intrigues further investigation of robots with better designed geometric patterns, which may lead to a higher-level control of swimming property.

5.4 Light controlled swimming

Here, a swimmer robot was made by using a black-colored plastic film, with a specific designed pattern, in order to achieve more control over the directional movement (Figure 23). The robot is composed of three edges/legs; all can be recognized by machine vision and selected to be exposed with laser beam irradiation. During the experiment, machine

vision subsystem tracked the robot and servo motor control subsystem was used to select the legs to be excited. When one side-leg of the robot was irradiated with laser beam, the whole body rotated clockwise or anti-clockwise (Table 6a,b). In case of the middle leg, the robot moved forward (Table 6c). Note that the locomotion was not entirely straight due to the fact that it is almost impossible to pin-point the laser beam exactly in the middle of the structure. This randomness in orientation during light-driven motion poses a hurdle in controlling robotic motion.

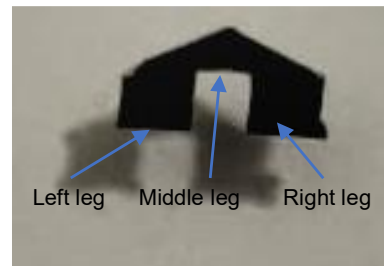


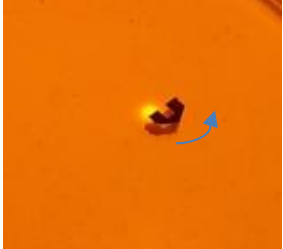





Figure 23. Structure of a light controlled swimming robot

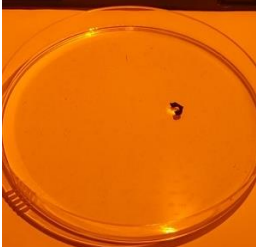
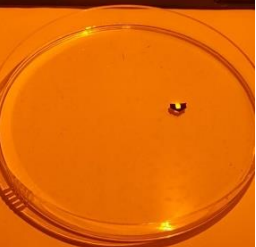
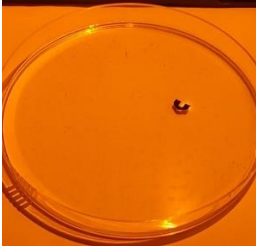
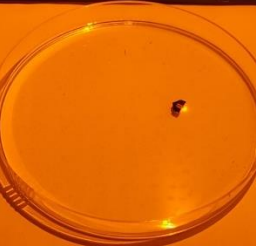
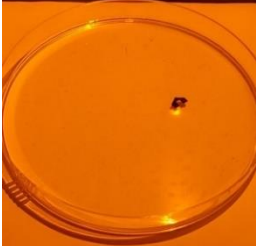
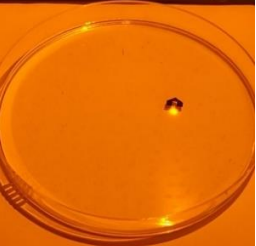
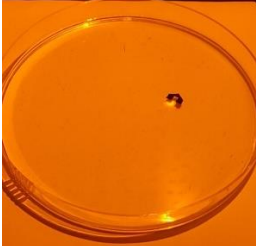
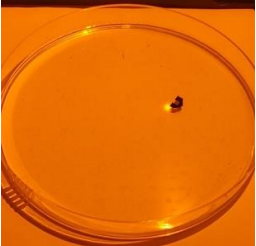
Table 6. Directional movements of a swimming robot.

| No | Initial stage | Locomotion due to the laser irradiation | Movement description |
|----|---|--|--|
| a |  |  | One side leg was irradiated with a laser beam which produced a clockwise rotation |
| b |  |  | Another side leg was irradiated with a laser beam which produced an anticlockwise rotation |
| c |  |  | Middle leg was irradiated with a laser beam which produced forward movement with a slight rotation |

Randomness also appears when illuminating the side edge of the robot to create rotating motion. Table 7 shows the experiment of a floating robot being excited on the side leg

with laser power of 700 mW. The robot rotated to clockwise directions with different angles (Table 7).

Table 7. *Rotational angle variations of a swimming robot.*

| Initial stage | Locomotion due to the laser irradiation | Approximate rotation angle(in degree) |
|---|--|---------------------------------------|
|  |  | 95 |
|  |  | 90 |
|  |  | 35 |
|  |  | 100 |

Though the laser power was fixed and continuous in that experiment, the rotational angle of this robot differs a lot. The randomness in light-driven rotation (excitation on the side-legs) and orientation during forward moving (excitation on the middle-leg) both affect the precision of light control in robotic motion. To solve this, reinforcement learning technique is developed to optimize the locomotive controllability, which will be introduced in the next Chapter.

6. REINFORCEMENT LEARNING IN LIGHT DRIVEN SWIMMING ROBOT (SIMULATIONS)

This chapter will focus on a light-driven floating robot's locomotion optimization by taking the randomness factor of movement into account. The aim is to develop an optimal control strategy to drive the robot to swim to a pre-selected location. To achieve this, reinforcement learning has been implemented because of its remarkable advantages in controlling robotic motion by learning from experiences without prior knowledge of the robot's performance. A variant of reinforcement learning, deep Q learning, is used to build a mapping from abstract observational data to actions for achieving optimal locomotion of the robot. Usually, reinforcement learning needs to deal with a large amount of training data before achieving optimal behaviors. However, collecting such amount of data is practically very challenging [23]. Thus, a simulator based on computer program is developed to train the deep Q network (DQN), before executing experiments on hardware. The simulator is tuned and trained with different randomness factors related to the floating robots to match the real-world scenario. The chapter will present the results of this simulation. Firstly, the RL environment and its components are described, then the agent and training parameters will be discussed. Finally, the simulation results will be presented.

6.1 Simulation environment

6.1.1 Action space and observational space

A swimming robot can exhibit three types of light-driven locomotion behaviors depending on which leg is subjected to laser irradiation. Figure 24a represents these movements: excitation on left/right leg yielding clockwise/anticlockwise rotation, and forward movement while pin-pointed the spot at the middle of the robot. Herein, the RL agent needs to train the object with three discrete actions, corresponding to the robot's clockwise rotation, forward movement and anticlockwise rotation, which are represented by numerical values 1, 2 and 3 in RL system. At each time, the agent selects an action among these three candidates, and selecting an action is referred to as one step in the simulation process.

After every action, the agent receives a set of data about observational states, which are associated with the information delivered by the simulator or the vision system. This data

includes the x- and y-axis values of robot position, target location, the distance between the robot and the target location, angle between the target and the robot as well as the orientation of the robot body (Figure 24b).

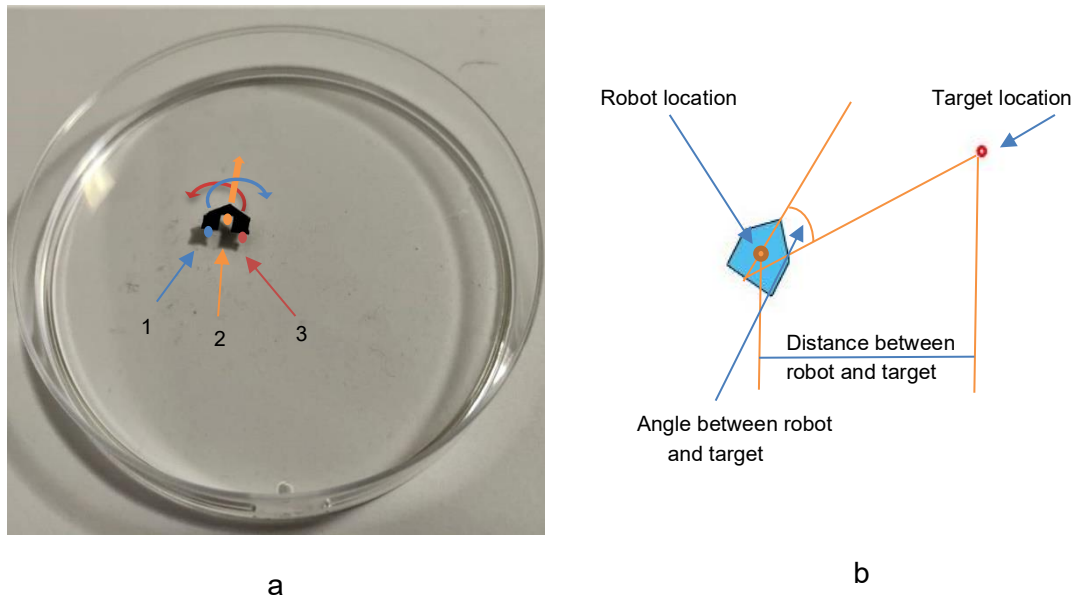


Figure 24. LCN robot. a. Action states. b. Observational states

6.1.2 Simulator

The swimming robot exhibits certain degrees of orientation with uncertainty and randomness in light-driven motion speed, as discussed in Chapter 5. Due to this reason, a simulator is modeled as a stochastic environment for learning the optimal policy using Deep Q learning. The simulator interface is illustrated in Figure 25 where the light blue color object represents the robot body, and the red dot indicates the targeted position where the robot is trained to approach. The whole area inside the red boundary is a workspace of the simulation serving as an effective region to execute robotic actions.

The simulator is associated with two methods: *position_control()* method and *measurement()* method. The *position_control()* method is responsible for determining the position and orientation of the robot after an action being taken by the agent and eventually updating the simulator. This method is controlled by assigning control strings 1, 2 or 3, which correspond to the robot's clockwise rotation, forward movement, anticlockwise rotation.

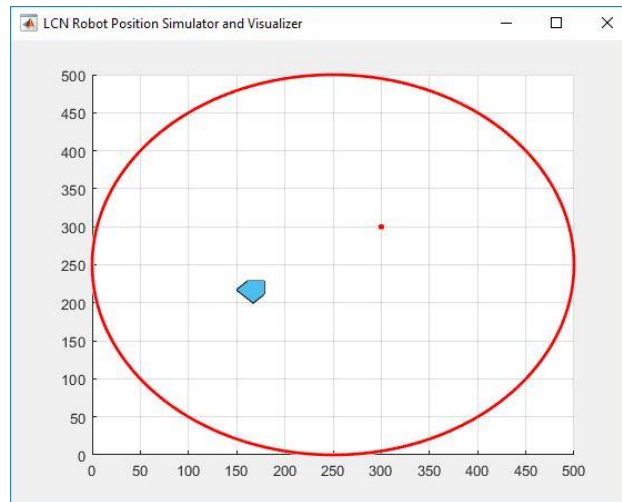


Figure 25. LCN robot position simulator and visualizer

After taking an action, the *measurement()* method returns the observational data. Besides, it also checks whether the robot has reached the target or not and the current position of the robot remains being inside the boundary circle or not. Initially, Deep Q learning was implemented on the simulator with simple case to get a simple solution. After that, the simulator is configured to mimic more complex situations by adding more parameters describing the randomness. These randomness effectors are tuned in the *position_control()* method. Finally, this simulator can provide an option for illustrating the whole trajectory of the robot, see details in Appendix C.

The simulator is designed in a way that it can still receive information from the vision system when being implemented in the real world. It plots the robot position, orientation and the target position in real-time, together with the trajectory of the robot. During the simulation, the simulator itself does not serve any observation data, as the data is directly fetched from the vision system. Besides the control strings used in the *position_control()* method also fully supports the previously developed servo motor control subsystem to steer the laser beam to the legs of the robot. Therefore, the training model in the simulator is effortlessly deployable in the real workspace (Appendix D, codes are available at this link: https://github.com/amankhan47/Aman_SPM_TUNI.git).

6.1.3 Reward function

The agent gets a reward for every action, which is calculated by a reward function. The reward function is defined by using distance and angle between the robot and the target location. The rewarding system is divided into two categories: (i) reward at the end of the episode (Algorithm 2) and (ii) reward for each action (Algorithm 3). When the robot reaches the target, touches the boundary, or reaches the maximum step, the episode ends. Based on each specific situation the agent gets a reward value. If the episode is

not a terminal episode then the reward function returns a reward value based on how good the action was, in terms of going towards or away from the target and maintaining small angle. The reward function inspires the agent to avoid the boundary and to take the forward steps to reach the target location by assigning higher rewards. Also, it motivates to maintain lower angular division during the locomotion with respect to the target. Besides, a constant negative reward is assigned after taking every single step, and this encourages the agent to reach the target by taking as little steps as possible. The aim of the agent is to accumulate maximum reward and reach the target, thus proper action knowledge is gained utilizing the developed reward function.

| |
|---|
| Algorithm 2: |
| IF episode is not in a terminal state Calculate the reward for the action has been taken (Algorithm 3) |
| ELSE IF robot is not inside the boundary Penalty, -100 |
| ELSE Reward for reaching target, 100 |
| Return reward |

| |
|---|
| Algorithm 3: |
| IF robot going towards the target Positive reward, 5 |
| ELSE IF going away from the target Penalty, -2 |
| ELSE not moved No reward, 0 |
| |
| IF low angle maintained between target and robot Reward for maintaining a low angle, 2 |
| ELSE Penalty for not maintaining a low angle, -10 |
| |
| Penalty for taking more step, -1 |
| RETURN total reward |

6.2 Setting up the agent and training

The agent utilizes deep Q-learning with experience replay algorithm. A critic network is built up to estimate the value function by conducting training. This network correlates rewards and delay rewards by receiving the observational states and action states. The critic network is configured with two input paths, the observational state path and action state path, and one output path named as common path (Figure 26). The observational

state path consists of an input layer and two fully connected convolution layer with an activation layer in between. The action state path has an input layer and a fully connected convolution layer. The common path combines the other two paths using an addition layer and connects to a fully connected convolution layer through an activation layer (see details in Table 8).

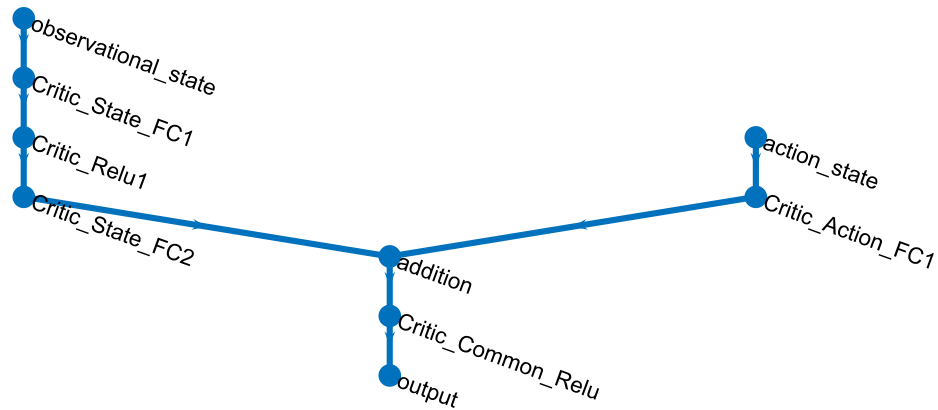


Figure 26. Architecture of the applied deep neural network

Table 8. Deep network layer description

| Name | Description |
|---------------------------------|--|
| Observational state path | |
| observational_state | input layer, input size:7x1x1 |
| Critic_State_FC1 | fully connected convolution layer, size:12 |
| Critic_Relu1 | activation layer, rectified linear unit |
| Critic_State_FC2 | fully connected convolution layer, size:12 |
| Action path | |
| Action_state | input layer, input size: 1x1x1 |
| Critic_Action_FC1 | fully connected convolution layer, size:12 |
| Common path | |
| addition | element-wise addition of 2 inputs |
| Critic_Common_Relu | activation layer, rectified linear unit |
| output | fully connected convolution layer, size:1 |

The RL system becomes ready for training after setting up the agent and training parameters. Every training session is divided into episodes, and in every episode the robot can take a maximum number of actions, *i.e.* steps. The maximum number of steps to reach the target is set to 20. The episode can end (a terminal state) when the robot touches the boundary line or the target location. After termination of an episode, the simulator restarts from the initial location to begin a new episode. The critic network employs gradient descent utilizing Adam optimizer with a learning rate of 0.01. The agent uses epsi-

lon greedy strategy to ensure a balance between exploration and exploitation. Depending on the value of epsilon, the agent decides which action should be taken, which may be a random one or the action with highest Q value. Initially, the agent collects experience data for later use in experience replay. The size of the replay memory is 10000, with a minibatch size of 32. Table 9 lists the used parameters for executing the training process. The best sequences of actions are saved for further use. Several training sessions were performed for achieving stable performance by utilizing the previous experience. Besides the simulator parameters were tuned to adapt the locomotion behavior of the swimming robot.

Table 9. Agent parameters for the Deep Q learning

| Parameters | Value |
|--------------------------|------------|
| Learning rate | 0.01 |
| Epsilon minimum | 0.01 |
| Epsilon decay | 0.005 |
| Discount factor | 0.9 |
| Experience buffer length | 10000 |
| Minimum batch size | 32 |
| Num steps to look ahead | 1 |
| Sample time | 1 |
| Target update method | periodic |
| Target update frequency | 4 |
| Target smooth factor | 1.0000e-03 |

6.3 Training without associating randomness in locomotion

The simulator was initialized with a fixed target location and simple movement behavior according to Table 10, without associating randomness by setting the *position_control()* method.

Table 10. Effect of selecting an action

| Selected action | Movement |
|-----------------|--|
| Actions 1 | Generate clockwise rotational movement, -25° |
| Actions 2 | Generate forward movement |
| Actions 3 | Generate anticlockwise rotational movement, 25° |

Figure 27 shows the training results of 500 episodes with about 5000 steps. The blue dots indicate the total reward for that particular episode, the red dots indicate average reward of the last five episodes and the green dots indicate the long term reward estimation(Q0), whose value is given by the calculation based on the initial simulator states and agent current condition.

According to the developed reward function, as shown in Figure 27, rewards of 100 or more were received by reaching the target position, but, a penalty of 100 or more was given if it passes the boundary. At the beginning of the training, the robot often crossed the boundary thus yielding low rewards. After one hundred episodes, it learned to reach the target and tried to achieve more rewards by spending fewer steps. After two hundred episodes the critic was able to correlate the estimated long-term reward(Q0) and the actual total reward with most of the episodes. In this case, the Q0 values conversed with the average reward values because there was no randomness factor included in the robot locomotion. After every training session, the episodes with reward values above 100 were saved for further training purposes.

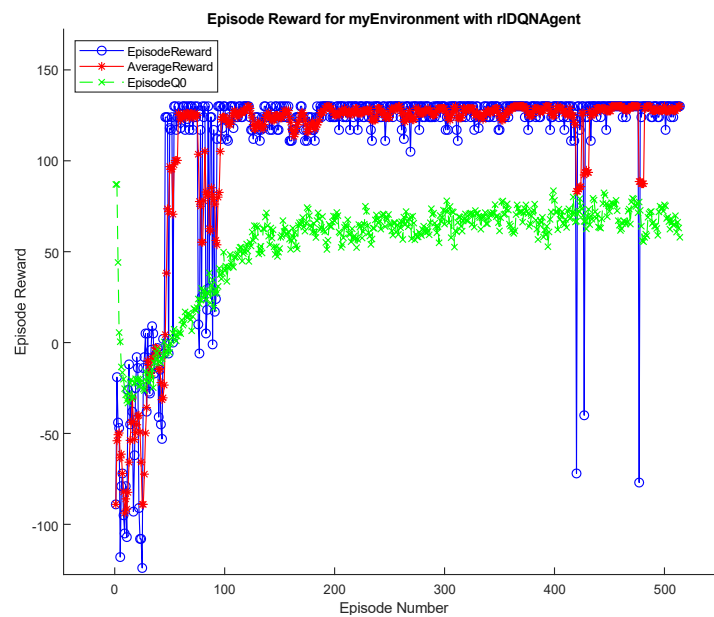


Figure 27. Training results without associating randomness in locomotion behavior with a fixed destination point

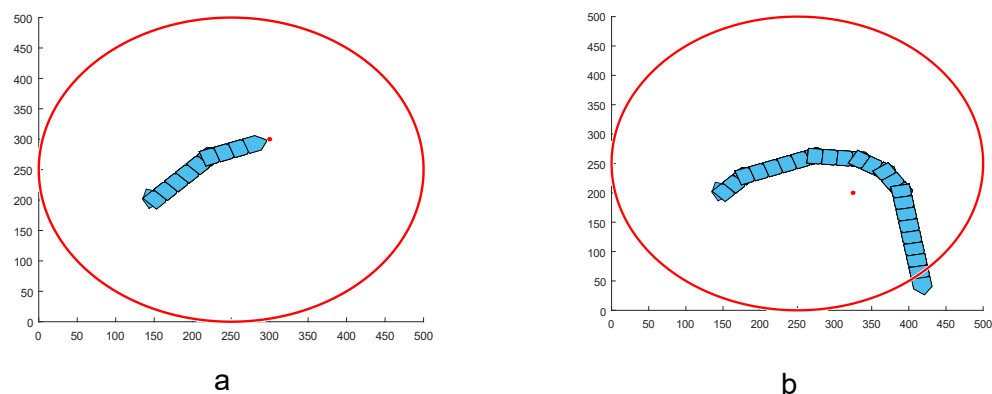


Figure 28. Evaluation of learned policy. a. Robot trajectory with the same target location used during training b. Robot trajectory with a different target location

The learned policy was evaluated with the simulator, by observing the robot trajectory for the fixed target location (used during the training session) (Figure 28a), and later on by changing to a different target location (Figure 28b). For the first case, the robot reached the target by spending eight steps and collected total rewards of 130. For a different target, the robot spent 25 steps and went outside the boundary. Importantly, the agent can move towards the target based on its learned skill.

6.4 Training approaching the real situation

The swimming robot always exhibits some degree of randomness in locomotion. To develop an optimal locomotion policy matching with the real scenario, different randomness factors were introduced in the simulator. The agent thus learned to adapt this randomness factors through different training sessions.

Introducing randomness factors into locomotion was done in three parts. The first part introduces the randomness factors in rotating movements. The robot rotates randomly in between 0 and 90° when the side legs are irradiated with a laser. Thus, a specific randomness factor has been added to the simulator to describe this random re-orientation, as shown in Table 11.

Table 11. *Effect of selecting an action for the first part*

| Selected action | Movement |
|------------------------|---|
| Actions 1 | generate random clockwise rotational movement which is calculated by $-90^\circ \times \text{random number in the interval of } (0,1)$ |
| Actions 2 | generate forward movement |
| Actions 3 | generate random anticlockwise rotational movement which is calculated by $90^\circ \times \text{random number in the interval of } (0,1)$ |

Then, the agent was trained with a fixed target location, and the training was conducted for 1200 episodes, together with about 13000 steps. The estimation value Q_0 correlated with most of the episode rewards, and most of the episodes got more than 100 rewards, *i.e.* the robot successfully reached the target (Figure 29). Then the effectiveness of the learned policy was accessed for two target points. The robot reached the fixed target (used in the training session) in eight steps and collected a total reward of 130 (Figure 30a). After learning, the robot is given a different target (other than the fixed target), and it was able to reach the target, by taking more steps and reaching a less reward (Figure 30b).

For the second part, the training was progressed with six different target locations for 2000 episodes with total step number of about 23000. The simulator changed the location every 99 episodes randomly which is also reflecting the training result (Figure 31). The evaluation of learned policy at different locations is shown in Figure 32 where the robot reaches the targets successfully.

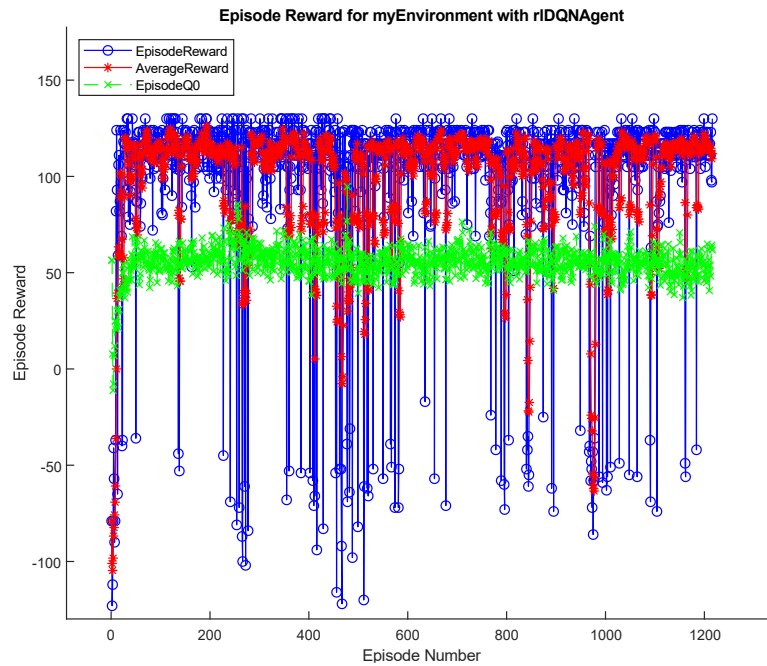


Figure 29. Training results with randomness in the rotational movement with a fixed destination point

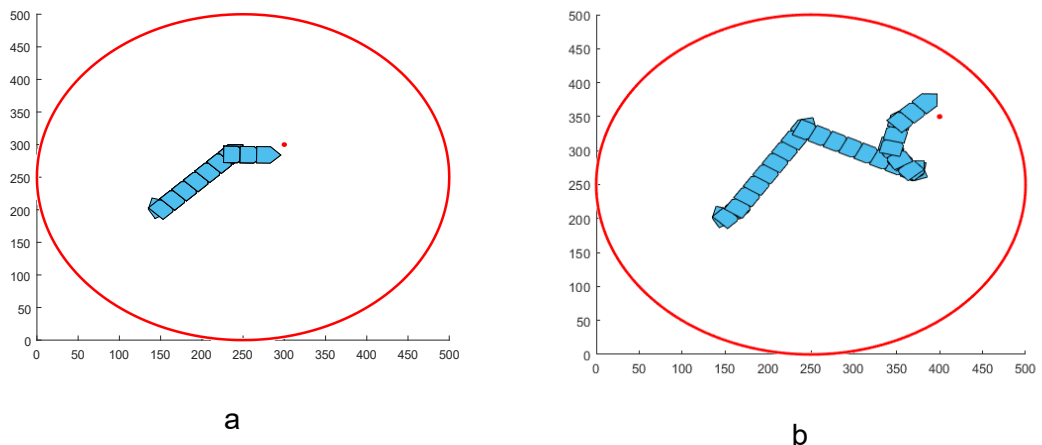


Figure 30. Evaluation of learned policy with randomness in the rotational movement. a. Robot trajectory with the same target location used during training b. Robot trajectory with a different target location

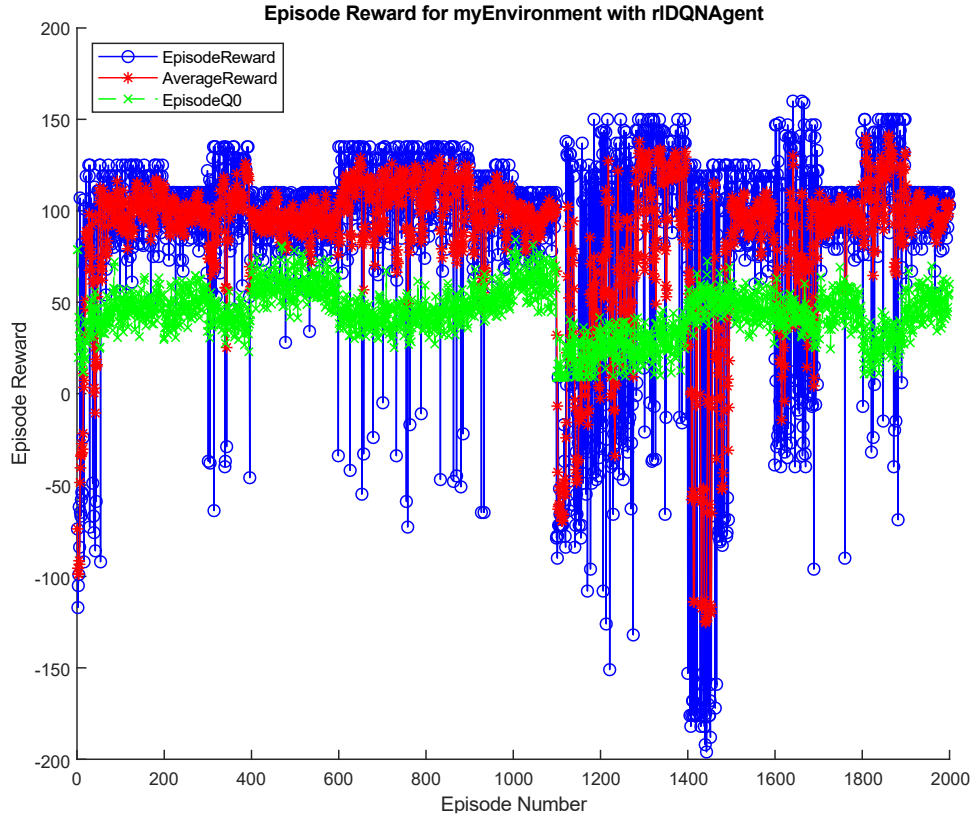


Figure 31. Training results with randomness in the rotational movement with different destination points

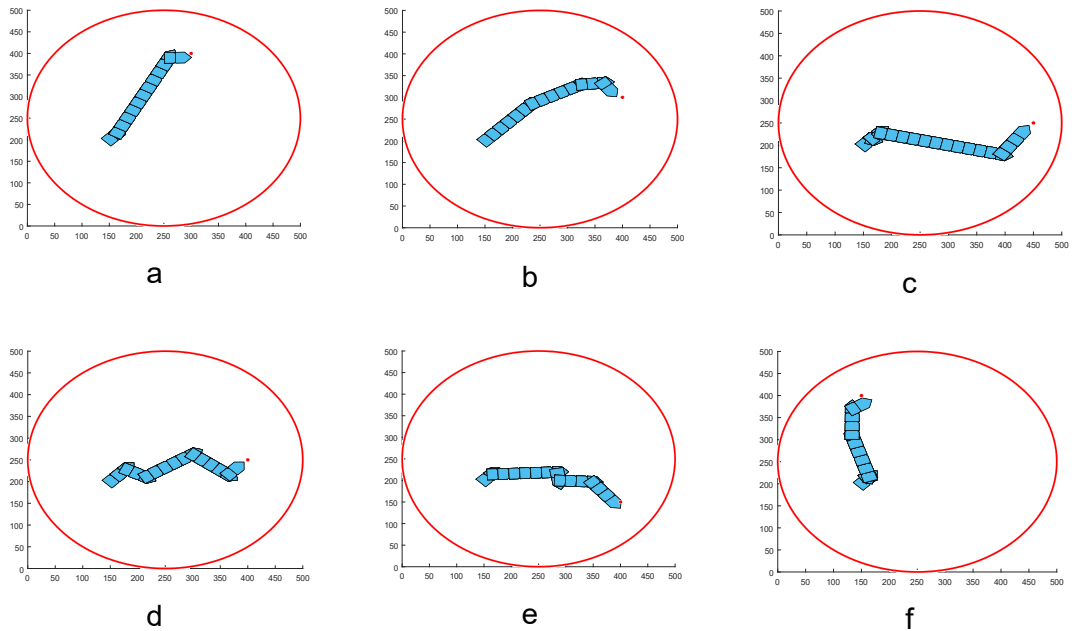


Figure 32. Evaluation of learned policy with different target locations and randomness in the rotational movement. a,b,c,d,e,f. Robot trajectory at different target location and robot successfully reaches the targets

For the third part, some random rotations were included in the robot's forward movements. This specific inclusion is due to the experimental observation that the forward movement of the robot is not entirely straight. Thus, randomness in actions given by using Table 12. Figure 33 shows the training results conducted in two sessions consisting 4500 episodes (70000 steps). After these training sessions, the robot can reach different targets successfully (Figure 34). As all the actions include randomness factor, the robot adapts suitable actions to reach the target point and may result in different trajectories for the same destination point.

Table 12. *Effect of selecting an action for the third part*

| Selected action | Movement |
|-----------------|---|
| Actions 1 | Generate random clockwise rotational movement which is calculated by $-90^\circ \times \text{random number}$ in the interval of (0,1) |
| Actions 2 | Generate forward movement with random clockwise or anticlockwise rotational movement, which is calculated by $(-45 \times \text{random number} + 45 \times \text{random number})$. The random number has an interval of (0,1). |
| Actions 3 | Generate random anticlockwise rotational movement which is calculated by $90^\circ \times \text{random number}$ in the interval of (0,1) |

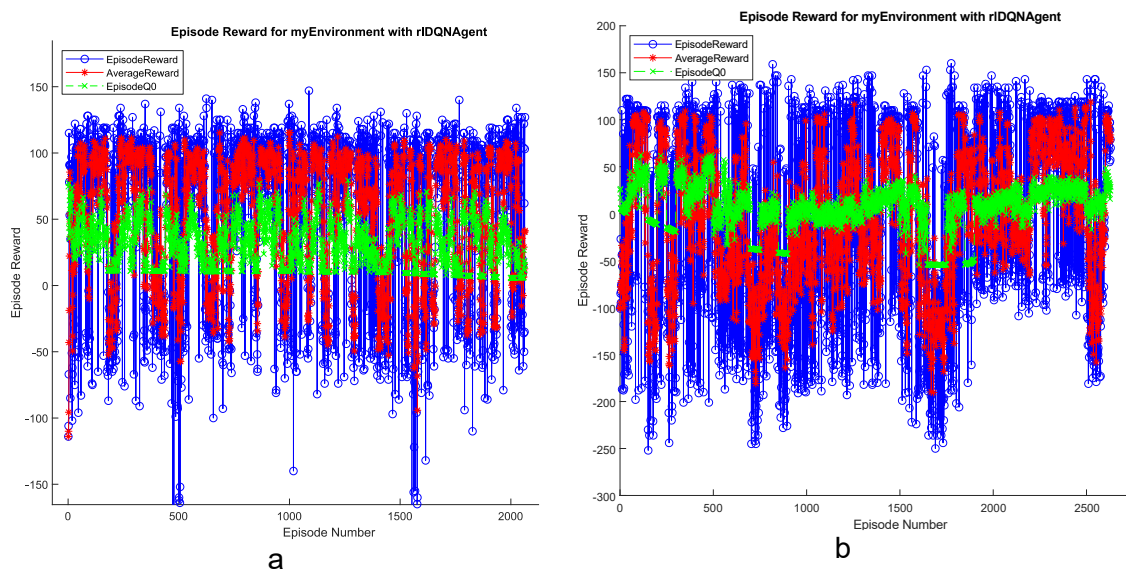


Figure 33. *Training results with randomness in every action with different random destination points (changing the points in every 50 episodes). a, b. Conducted training in two sessions*

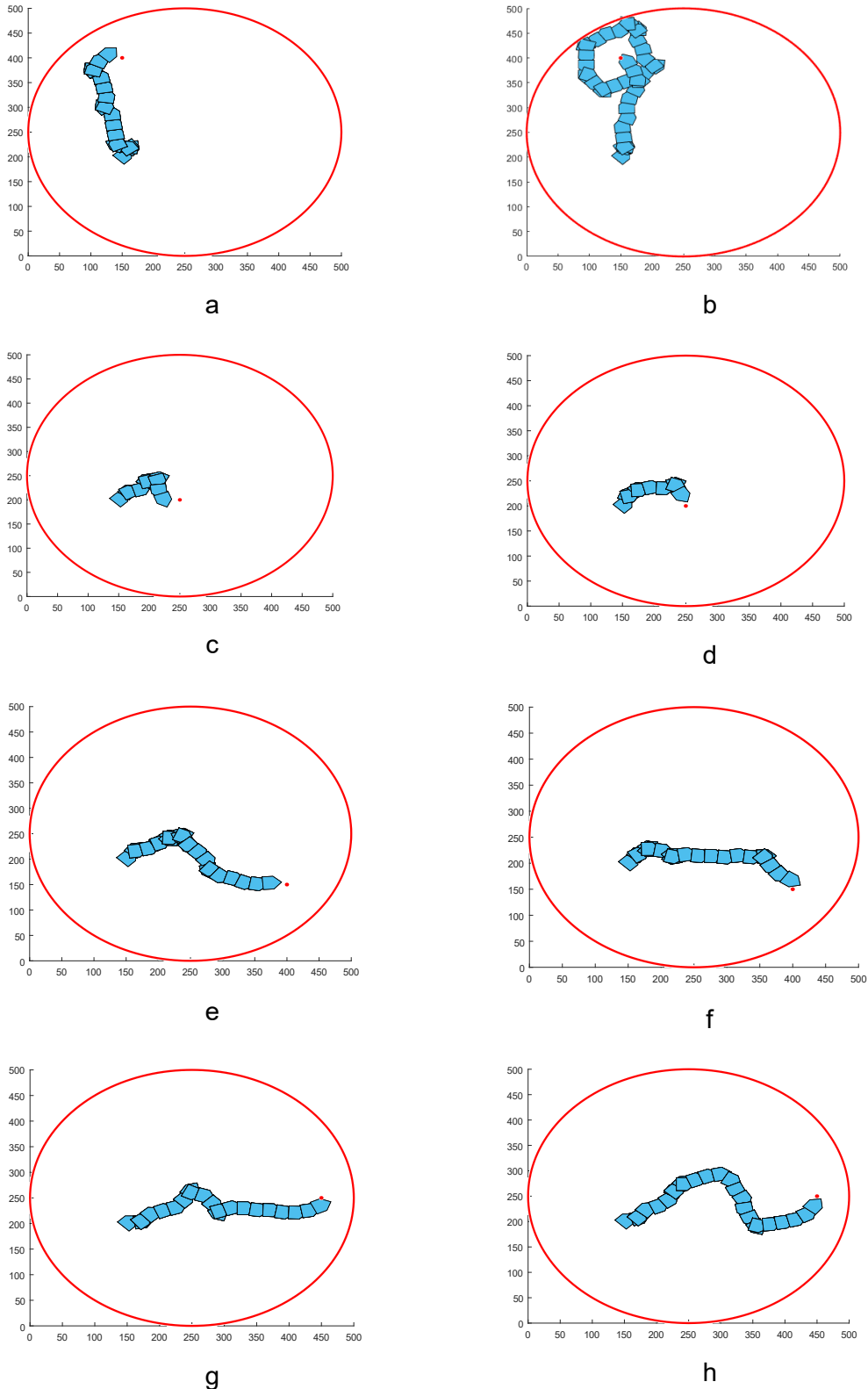


Figure 34. Evaluation of learned policy with different target locations and randomness factors included in all actions. At the same target, robot successfully reached using different trajectories. The same target pairs are (a,b);(c,d),(e,f);(g,h)

Gradually all the randomness factors were introduced in these training processes. In the last part of the training, all the actions include randomness factors in order to resemble the swimming robot locomotion. Due to the randomness in every action, the robot chooses the best action (possibly) depending on the latest situation of the robot. The aim is to promote the directional movement of the swimmer with orientation towards the target point. Overall, the agent attempts to move the robot towards different destination points while maintaining small angle with respect to the destination point and spending fewer steps. After all these training sessions (approximately 120 000 steps), the trajectory results confirm that the deep Q learning agent is successful, and an optimal control strategy to reach different target locations is developed.

7. CONCLUSIONS AND OUTLOOK

This Thesis study tried to develop an optical control method to enable directional locomotion of small-scale photoresponsive robots as well as an optimal control strategy for locomotion possibly that can be used in future soft robotics.

The developed robot tracking system provided an efficient detection and analysis platform for the small-scale photoresponsive robots through laser steering to the detected features. The system was able to move laser efficiently by changing some commands, as shown in Chapters 4 and 5. This system assisted in realizing different factors related to LCN robots' locomotion (walking and swimming). Among different types of photoresponsive robots, the swimmer robot exhibited potential directional movement, albeit with some degree of uncertainty and randomness in their locomotion.

This Thesis presented a new approach for controlling locomotion of photoresponsive swimmer robots by using deep Q learning, in order to optimize the moving efficiency to reach a target. A simulator was developed to mimic the uncertain and random behavior of those robots and used to collect a large number of training samples. The simulator allowed to choose the way of exploring the environment and enabled the RL agent to better understand the environment and decide the best actions. Gradual progress had been made in the simulator where the agent learned from the fundamental to complex movement behavior. Thousands of episodes had been conducted to develop an optimal control policy where the robot spent fewer steps and took the possible shortest path (considering all the randomness in every movement) to reach different target destinations.

This work has successfully taken a step towards controlling the autonomous locomotion of light-responsive soft robots. Furthermore, the agent can be trained as well as the learned policy can be applied in the real environment because the entire system is developed in a way that it fully supports and correlates the robot tracking system. Thus, the future step will be to deploy the learned optimal control policy with actual swimmer robots in the real environment.

The developed robot tracking system can be used in different experiments to recognize the different features of photoresponsive materials and robots. The machine vision program uses a template matching technique, which enables affluent detection of desired features. Besides laser steering system gives sequential options to steer the laser beam

automatically to detected features. This system already has been used for conducting several experiments related to light-responsive materials and robots.

Application of deep reinforcement learning in soft photoresponsive robotics has many potentials because of dynamic adaptation to the environment without the information of structural configuration and constituent material properties as well as robotic mechanics. This work can be extended to a real environment for locomotion control, obstacle avoidance, micro-object delivery, and multiple robot control to accomplish collaborative tasks. Also, the whole system can be used for photoresponsive robots of different types and designs with minor modifications because of straightforward feature detection, efficient laser steering, and facile insertion of different movement patterns in the simulator to train the reinforcement learning agent. This approach of controlling robot locomotion can bridge and make the path to locomotion control of different light-driven robots.

REFERENCES

- [1] D. Rus and M. T. Tolley, "Design, fabrication and control of soft robots," *Nature*, vol. 521, no. 7553. Nature Publishing Group, pp. 467–475, 27-May-2015.
- [2] H. Abramovich, *Intelligent Materials and Structures*. De Gruyter, 2016.
- [3] S. Maeda, Y. Hara, T. Sakai, R. Yoshida, and S. Hashimoto, "Self-Walking Gel," *Adv. Mater.*, vol. 19, no. 21, pp. 3480–3484, Nov. 2007.
- [4] S. Tottori, L. Zhang, F. Qiu, K. K. Krawczyk, A. Franco-Obregón, and B. J. Nelson, "Magnetic Helical Micromachines: Fabrication, Controlled Swimming, and Cargo Transport," *Adv. Mater.*, vol. 24, no. 6, pp. 811–816, Feb. 2012.
- [5] C. Niezrecki, D. Brei, S. Balakrishnan, and A. Moskalik, "Piezoelectric Actuation: State of the Art," *Shock Vib. Dig.*, Jan. 2001.
- [6] L. Hines, K. Petersen, G. Z. Lum, and M. Sitti, "Soft Actuators for Small-Scale Robotics," *Advanced Materials*, vol. 29, no. 13, p. 1603483, Apr-2017.
- [7] O. Wani, "Bioinspired Light Robots from Liquid Crystal Networks." Tampere University, 2019.
- [8] O. M. Wani, H. Zeng, and A. Priimagi, "A light-driven artificial flytrap," *Nat. Commun.*, vol. 8, no. 1, p. 15546, Aug. 2017.
- [9] I. Y. Galaev and B. Mattiasson, "'Smart' polymers and what they could do in biotechnology and medicine," *Trends in Biotechnology*, vol. 17, no. 8, pp. 335–340, 01-Aug-1999.
- [10] H. Zeng, P. Wasylczyk, D. S. Wiersma, and A. Priimagi, "Light Robots: Bridging the Gap between Microrobotics and Photomechanics in Soft Materials," *Adv. Mater.*, vol. 30, no. 24, p. 1703554, Jun. 2018.
- [11] F. Ilievski, A. D. Mazzeo, R. F. Shepherd, X. Chen, and G. M. Whitesides, "Soft Robotics for Chemists," *Angew. Chemie Int. Ed.*, vol. 50, no. 8, pp. 1890–1895, Feb. 2011.
- [12] O. M. Wani, H. Zeng, and A. Priimagi, "A light-driven artificial flytrap," *Nat. Commun.*, vol. 8, no. 1, p. 15546, Aug. 2017.
- [13] H. Zeng, M. Lahikainen, O. M. Wani, A. Berdin, and A. Priimagi, "Liquid Crystal Polymer Networks and Elastomers for Light-Fueled Robotics," in *Photoactive Functional Soft Materials*, Wiley, 2019, pp. 197–226.
- [14] T. (Timothy) White, *Photomechanical materials, composites, and systems: wireless transduction of light into work*. .
- [15] R. J. Webster and B. A. Jones, "Design and Kinematic Modeling of Constant Curvature Continuum Robots: A Review," *Int. J. Rob. Res.*, vol. 29, no. 13, pp. 1661–1683, Nov. 2010.

- [16] H. Zhang, R. Cao, S. Zilberstein, F. Wu, and X. Chen, "Toward Effective Soft Robot Control via Reinforcement Learning."
- [17] P. Hyatt, D. Wingate, and M. D. Killpack, "Model-based control of soft actuators using learned non-linear discrete-time models," *Front. Robot. AI*, vol. 6, no. APR, 2019.
- [18] S. Bhagat, H. Banerjee, Z. T. H. Tse, and H. Ren, "Deep reinforcement learning for soft, flexible robots: Brief review with impending challenges," *Robotics*, vol. 8, no. 1. MDPI, p. 4, 18-Jan-2019.
- [19] A. El Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," in *IS and T International Symposium on Electronic Imaging Science and Technology*, 2017, pp. 70–76.
- [20] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. 2008.
- [21] G. W. B, L. Ren, and J. S. Dai, "Toward Effective Soft Robot Control via Reinforcement Learning," vol. 1, pp. 71–83, 2017.
- [22] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," Dec. 2013.
- [23] P. Yue, J. Xin, H. Zhao, D. Liu, M. Shan, and J. Zhang, "Experimental Research on Deep Reinforcement Learning in Autonomous navigation of Mobile Robot," *2019 14th IEEE Conf. Ind. Electron. Appl.*, pp. 1612–1616, 2019.
- [24] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation."
- [25] J. Xin, H. Zhao, D. Liu, and M. Li, "Application of deep reinforcement learning in mobile robot path planning," in *Proceedings - 2017 Chinese Automation Congress, CAC 2017*, 2017, vol. 2017-Janua, pp. 7112–7116.
- [26] H. Sasaki, T. Horiuchi, and S. Kato, "A study on vision-based mobile robot learning by deep Q-network," *2017 56th Annu. Conf. Soc. Instrum. Control Eng. Japan, SICE 2017*, vol. 2017-Novem, pp. 799–804, 2017.
- [27] Y. Cheng and W. Zhang, "Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels," *Neurocomputing*, vol. 272, pp. 63–73, 2018.
- [28] S. James and E. Johns, "3D Simulation for Robot Arm Control with Deep Q-Learning," 2016.
- [29] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, "Towards vision-based deep reinforcement learning for robotic motion control," *Australas. Conf. Robot. Autom. ACRA*, 2015.
- [30] F. Reinitzer, "Beiträge zur Kenntniss des Cholesterins," *Monatshefte für Chemie - Chem. Mon.*, vol. 9, no. 1, pp. 421–441, Dec. 1888.
- [31] A. Priimagi, C. J. Barrett, and A. Shishido, "Recent twists in photoactuation and photoalignment control," *J. Mater. Chem. C*, vol. 2, no. 35, pp. 7155–7162, 2014.
- [32] J. W. G. Goodby, *Handbook of liquid crystals*. .

- [33] T. J. White and D. J. Broer, "Programmable and adaptive mechanics with liquid crystal polymer networks and elastomers Preparation and properties of LCEs and LCNs," 2015.
- [34] S. Palagi *et al.*, "Locomotion of light-driven soft microrobots through a hydrogel via local melting," in *2017 International Conference on Manipulation, Automation and Robotics at Small Scales (MARSS)*, 2017, pp. 1–5.
- [35] T. J. White, "Photomechanical effects in liquid crystalline polymer networks and elastomers," *J. Polym. Sci. Part B Polym. Phys.*, vol. 56, no. 9, pp. 695–705, May 2018.
- [36] H. Yu and T. Ikeda, "Photocontrollable Liquid-Crystalline Actuators," *Adv. Mater.*, vol. 23, no. 19, pp. 2149–2180, May 2011.
- [37] E. Merino and M. Ribagorda, "Control over molecular motion using the cis – trans photoisomerization of the azo group," *Beilstein J. Org. Chem.*, vol. 8, no. 1, pp. 1071–1090, Jul. 2012.
- [38] H. Yu, "Recent advances in photoresponsive liquid-crystalline polymers containing azobenzene chromophores," *J. Mater. Chem. C*, vol. 2, no. 17, pp. 3047–3054, Apr. 2014.
- [39] H. M. D. Bandara and S. C. Burdette, "Photoisomerization in different classes of azobenzene," *Chem. Soc. Rev.*, vol. 41, no. 5, pp. 1809–1825, Feb. 2012.
- [40] Z. Mahimwalla, K. G. Yager, J. I. Mamiya, A. Shishido, A. Priimagi, and C. J. Barrett, "Azobenzene photomechanics: Prospects and potential applications," *Polym. Bull.*, vol. 69, no. 8, pp. 967–1006, Nov. 2012.
- [41] T. Ikeda, "Photomodulation of liquid crystal orientations for photonic applications," *J. Mater. Chem.*, vol. 13, no. 9, p. 2037, Aug. 2003.
- [42] J. García-Amorós and D. Velasco, "Recent advances towards azobenzene-based light-driven real-time information-transmitting materials," *Beilstein J. Org. Chem.*, vol. 8, no. 1, pp. 1003–1017, Jul. 2012.
- [43] C. Knie *et al.*, "*ortho* -Fluoroazobenzenes: Visible Light Switches with Very Long-Lived *Z* Isomers," *Chem. - A Eur. J.*, vol. 20, no. 50, pp. 16492–16501, Dec. 2014.
- [44] J. Calbo, C. E. Weston, A. J. P. White, H. S. Rzepa, J. Contreras-García, and M. J. Fuchter, "Tuning azoheteroarene photoswitch performance through heteroaryl design," *J. Am. Chem. Soc.*, vol. 139, no. 3, pp. 1261–1274, 2017.
- [45] S. Iamsaard, E. Anger, S. J. Aßhoff, A. Depauw, S. P. Fletcher, and N. Katsonis, "Fluorinated Azobenzenes for Shape-Persistent Liquid Crystal Polymer Networks," *Angew. Chemie Int. Ed.*, vol. 55, no. 34, pp. 9908–9912, Aug. 2016.
- [46] D. Bléger and S. Hecht, "Visible-Light-Activated Molecular Switches," *Angew. Chemie Int. Ed.*, vol. 54, no. 39, pp. 11338–11349, Sep. 2015.
- [47] D. J. Broer, G. P. Crawford, and S. Žumer, *Cross-linked liquid crystalline systems: From rigid polymer networks to elastomers*. CRC Press, 2011.
- [48] Y. Hu, Z. Li, T. Lan, and W. Chen, "Photoactuators for Direct Optical-to-Mechanical Energy Conversion: From Nanocomponent Assembly to Macroscopic

- Deformation," *Adv. Mater.*, vol. 28, no. 47, pp. 10548–10556, Dec. 2016.
- [49] H. Zeng, P. Wasylczyk, C. Parmeggiani, D. Martella, M. Burrese, and D. S. Wiersma, "Light-Fueled Microscopic Walkers," *Adv. Mater.*, vol. 27, no. 26, pp. 3883–3887, Jul. 2015.
- [50] D. Martella, S. Nocentini, D. Nuzhdin, C. Parmeggiani, and D. S. Wiersma, "Photonic Microhand with Autonomous Action," *Adv. Mater.*, vol. 29, no. 42, p. 1704047, Nov. 2017.
- [51] A. H. Gelebart, G. Vantomme, E. W. Meijer, and D. J. Broer, "Mastering the Photothermal Effect in Liquid Crystal Networks: A General Approach for Self-Sustained Mechanical Oscillators," *Adv. Mater.*, vol. 29, no. 18, p. 1606712, May 2017.
- [52] L.-X. Guo *et al.*, "A calamitic mesogenic near-infrared absorbing croconaine dye/liquid crystalline elastomer composite," *Chem. Sci.*, vol. 7, no. 7, pp. 4400–4406, Jun. 2016.
- [53] M. Lahikainen, H. Zeng, and A. Priimagi, "Reconfigurable photoactuator through synergistic use of photochemical and photothermal effects," *Nat. Commun.*, vol. 9, no. 1, p. 4148, Dec. 2018.
- [54] Y. Ji, J. E. Marshall, and E. M. Terentjev, "Nanoparticle-Liquid Crystalline Elastomer Composites," *Polymers (Basel)*, vol. 4, pp. 316–340, 2012.
- [55] R. Yin *et al.*, "Can sunlight drive the photoinduced bending of polymer films?," *J. Mater. Chem.*, vol. 19, no. 20, p. 3141, May 2009.
- [56] F. Cheng, Y. Zhang, R. Yin, and Y. Yu, "Visible light induced bending and unbending behavior of crosslinked liquid-crystalline polymer films containing azotolane moieties," *J. Mater. Chem.*, vol. 20, no. 23, p. 4888, Jun. 2010.
- [57] W. Wu, L. Yao, T. Yang, R. Yin, F. Li, and Y. Yu, "NIR-Light-Induced Deformation of Cross-Linked Liquid-Crystal Polymers Using Upconversion Nanophosphors," *J. Am. Chem. Soc.*, vol. 133, no. 40, pp. 15810–15813, Oct. 2011.
- [58] Y. Yu, M. Nakano, and T. Ikeda, "Directed bending of a polymer film by light," *Nature*, vol. 425, no. 6954, pp. 145–145, Sep. 2003.
- [59] T. Ikeda, M. Nakano, Y. Yu, O. Tsutsumi, and A. Kanazawa, "Anisotropic Bending and Unbending Behavior of Azobenzene Liquid-Crystalline Gels by Light Exposure," *Adv. Mater.*, vol. 15, no. 3, pp. 201–205, Feb. 2003.
- [60] H. Zeng, O. M. Wani, P. Wasylczyk, R. Kaczmarek, and A. Priimagi, "Self-Regulating Iris Based on Light-Actuated Liquid Crystal Elastomer," *Adv. Mater.*, vol. 29, no. 30, p. 1701814, Aug. 2017.
- [61] S. Iamsaard, E. Anger, S. J. Aßhoff, A. Depauw, S. P. Fletcher, and N. Katsonis, "Fluorinated Azobenzenes for Shape-Persistent Liquid Crystal Polymer Networks," *Angew. Chemie - Int. Ed.*, vol. 55, no. 34, pp. 9908–9912, 2016.
- [62] A. H. Gelebart, M. Mc Bride, A. P. H. J. Schenning, C. N. Bowman, and D. J. Broer, "Photoresponsive Fiber Array: Toward Mimicking the Collective Motion of Cilia for Transport Applications," *Adv. Funct. Mater.*, vol. 26, no. 29, pp. 5322–5327, Aug. 2016.

- [63] C. L. van Oosten, C. W. M. Bastiaansen, and D. J. Broer, "Printed artificial cilia from liquid-crystal network actuators modularly driven by light," *Nat. Mater.*, vol. 8, no. 8, pp. 677–682, Aug. 2009.
- [64] Y. Bellouard, "Applied Physics for Microrobotics," in *Microrobotics*, 2009, pp. 113–158.
- [65] I. Shimoyama, "Scaling in microrobots," in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, vol. 2, pp. 208–211.
- [66] J. W. Hermanson, "Principles of Animal Locomotion," *J. Mammal.*, vol. 85, no. 3, pp. 584–584, Jun. 2004.
- [67] J. N. Israelachvili, *Intermolecular and surface forces*. Academic Press, 2011.
- [68] N. R. Tas, C. Gui, and M. Elwenspoek, "Static friction in elastic adhesion contacts in MEMS," *J. Adhes. Sci. Technol.*, vol. 17, no. 4, pp. 547–561, Jan. 2003.
- [69] E. M. Purcell, "Life at low Reynolds number," *Am. J. Phys.*, vol. 45, no. 1, pp. 3–11, Jan. 1977.
- [70] M. Salta *et al.*, "Designing biomimetic antifouling surfaces," *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, vol. 368, no. 1929, pp. 4729–4754, Oct. 2010.
- [71] T. Ikeda, J. Mamiya, and Y. Yu, "Photomechanics of Liquid-Crystalline Elastomers and Other Polymers," *Angew. Chemie Int. Ed.*, vol. 46, no. 4, pp. 506–528, Jan. 2007.
- [72] M. Yamada *et al.*, "Photomobile Polymer Materials: Towards Light-Driven Plastic Motors," *Angew. Chemie Int. Ed.*, vol. 47, no. 27, pp. 4986–4988, Jun. 2008.
- [73] M. Yamada *et al.*, "Photomobile polymer materials—various three-dimensional movements," *J. Mater. Chem.*, vol. 19, no. 1, pp. 60–62, Dec. 2009.
- [74] R. R. Kohlmeyer and J. Chen, "Wavelength-Selective, IR Light-Driven Hinges Based on Liquid Crystalline Elastomer Composites," *Angew. Chemie Int. Ed.*, vol. 52, no. 35, pp. 9234–9237, Aug. 2013.
- [75] H. Zeng, P. Wasylczyk, C. Parmeggiani, D. Martella, M. Burresti, and D. S. Wiersma, "Artificial Muscle: Light-Fueled Microscopic Walkers (Adv. Mater. 26/2015)," *Adv. Mater.*, vol. 27, no. 26, pp. 3842–3842, Jul. 2015.
- [76] M. Rogóż, H. Zeng, C. Xuan, D. S. Wiersma, and P. Wasylczyk, "Light-Driven Soft Robot Mimics Caterpillar Locomotion in Natural Scale," *Adv. Opt. Mater.*, vol. 4, no. 11, pp. 1689–1694, 2016.
- [77] C. Huang, J.-A. Lv, X. Tian, Y. Wang, Y. Yu, and J. Liu, "Miniaturized Swimming Soft Robot with Complex Movement Actuated and Controlled by Remote Light Signals OPEN," *Nat. Publ. Gr.*, 2015.
- [78] S. Palagi *et al.*, "Structured light enables biomimetic swimming and versatile locomotion of photoresponsive soft microrobots," *Nat. Mater.*, vol. 15, no. 6, pp. 647–653, 2016.
- [79] H. Zeng, O. M. Wani, P. Wasylczyk, and A. Priimagi, "Light-Driven, Caterpillar-

- Inspired Miniature Inching Robot,” *Macromol. Rapid Commun.*, vol. 39, no. 1, p. 1700224, Jan. 2018.
- [80] B. G. Batchelor, *Machine vision handbook*. 2012.
- [81] W. Liu, Z. Jing, G. D’Eleuterio, W. Chen, T. Yang, and H. Pan, “Shape Memory Alloy Driven Soft Robot Design and Position Control Using Continuous Reinforcement Learning,” pp. 124–130, 2019.
- [82] E. R. Davies, *Machine Vision: Theory, Algorithms, Practicalities (Signal Processing and its Applications)*. New York: Academic Press, 2005.
- [83] A. Hornberg, *Handbook of Machine Vision*. Wiley-VCH, 2007.
- [84] A. Hornberg, *Handbook of Machine and Computer Vision: the Guide for Developers and Users*. John Wiley & Sons, Incorporated, 2017.
- [85] C. Steger, M. Ulrich, and C. Wiedemann, *Machine Vision Algorithms and Applications*. 2018.
- [86] “Filtering in Machine Vision | Edmund Optics.” [Online]. Available: <https://www.edmundoptics.eu/resources/application-notes/imaging/filtering-in-machine-vision/>. [Accessed: 23-Oct-2019].
- [87] “The Advantages of Telecentricity | Edmund Optics.” [Online]. Available: <https://www.edmundoptics.eu/resources/application-notes/imaging/advantages-of-telecentricity/>. [Accessed: 24-Oct-2019].
- [88] A. El Gamal and H. Eltoukhy, “CMOS image sensors,” *IEEE Circuits Devices Mag.*, vol. 21, no. 3, pp. 6–20, May 2005.
- [89] G. C. Holst, T. S. Lomheim, and Society of Photo-optical Instrumentation Engineers, *CMOS/CCD sensors and camera systems*. .
- [90] “What are the benefits of CMOS based machine vision cameras vs CCD?” [Online]. Available: <https://www.1stvision.com/machine-vision-solutions/2019/07/benefits-of-cmos-based-machine-vision-cameras-vs-ccd.html>. [Accessed: 24-Oct-2019].
- [91] Z. Illes *et al.*, *Computer Vision: Algorithms and Applications*, vol. 35, no. 12. 2005.
- [92] P. Corke, *Image formation*, vol. 73. 2011.
- [93] V. Misrai, A. de la Taille, and M. Rouprêt, “In Peer (Artificial Intelligence) Review We Trust,” *Eur. Urol.*, vol. 76, no. 1, pp. 133–135, Jul. 2019.
- [94] D. T. Hogarty, D. A. Mackey, and A. W. Hewitt, “Current state and future prospects of artificial intelligence in ophthalmology: a review,” *Clin. Experiment. Ophthalmol.*, vol. 47, no. 1, pp. 128–139, Jan. 2019.
- [95] J. (Consultant) Patterson and A. Gibson, *Deep learning: a practitioner’s approach*. .
- [96] S. Russell, *Artificial Intelligence : a Modern Approach*. Pearson Education Limited, 2016.

- [97] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.
- [98] C. McCue, *Information Security Analytics*. Elsevier, 2015.
- [99] G. Shobha and S. Rangaswamy, "Machine Learning," *Handb. Stat.*, vol. 38, pp. 197–228, Jan. 2018.
- [100] *Data Mining and Predictive Analysis*. Elsevier, 2015.
- [101] J. Si, "Direct Learning by Reinforcement," *Electr. Eng. Handb.*, pp. 1151–1159, Jan. 2005.
- [102] X. Xue, Z. Li, D. Zhang, and Y. Yan, "A Deep Reinforcement Learning Method for Mobile Robot Collision Avoidance based on Double DQN," in *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*, 2019, pp. 2131–2136.
- [103] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [104] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.
- [105] R. Buyya *et al.*, "Deep Learning and Its Parallelization," *Big Data*, pp. 95–118, Jan. 2016.
- [106] "Reinforcement Learning with MATLAB," *The MathWorks, Inc.*, 2019. [Online]. Available: <https://www.mathworks.com>. [Accessed: 18-Sep-2019].
- [107] F. Tan, P. Yan, and X. Guan, "Deep Reinforcement Learning: From Q-Learning to Deep Q-Learning," 2017, pp. 475–483.
- [108] N. Kwak, S. Yoon, and K. Roh, "Learning Motion Policy for Mobile Robots Using Deep Q-Learning," *Proc. - 2017 Int. Conf. Comput. Sci. Comput. Intell. CSCI 2017*, pp. 805–810, 2018.
- [109] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A Brief Survey of Deep Reinforcement Learning," pp. 1–16.
- [110] P. S. Thomas, "Bias in natural actor-critic algorithms," *31st Int. Conf. Mach. Learn. ICML 2014*, vol. 1, no. 11, pp. 693–700, 2014.
- [111] F. Tan, P. Yan, and X. Guan, "Deep Reinforcement Learning: From Q-Learning to Deep Q-Learning," Springer, Cham, 2017, pp. 475–483.
- [112] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, 1992.
- [113] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.
- [114] J. N. Tsitsiklis and B. Van Roy, "Analysis of temporal-difference learning with function approximation," in *Advances in Neural Information Processing Systems*, 1997, pp. 1075–1081.
- [115] "Carl Zeiss Microscopy GmbH." [Online]. Available: <https://www.micro->

shop.zeiss.com/en/de/. [Accessed: 29-Oct-2019].

- [116] “CMOS Cameras: USB 2.0 and USB 3.0.” [Online]. Available: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=4024. [Accessed: 29-Oct-2019].
- [117] “Zoom Macro Lenses | Navitar Optical Solutions.” [Online]. Available: <https://navitar.com/products/imaging-optics/low-magnification-video/zoom-macro/>. [Accessed: 29-Oct-2019].
- [118] “Precision Kinematic Mirror Mounts.” [Online]. Available: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=3. [Accessed: 29-Oct-2019].
- [119] “LD PUMPED ALL-SOLID-STATE GREEN LASER.” [Online]. Available: <http://www.cnilaser.com/PDF/MGL-F-532.pdf>. [Accessed: 29-Oct-2019].
- [120] “1/2" (12 mm or 13 mm) Travel Motorized Actuators.” [Online]. Available: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=1882. [Accessed: 29-Oct-2019].
- [121] “Kinesis® K-Cube™ Brushed DC Servo Motor Controller.” [Online]. Available: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=2419&pn=KDC101. [Accessed: 29-Oct-2019].
- [122] J. Caja, E. Gómez, P. Maresca, and M. Berzal, “Development of a calibration model for optical measuring machines,” *Procedia Eng.*, vol. 63, pp. 225–233, 2013.
- [123] J. Weng, P. Coher, and M. Herniou, “Camera Calibration with Distortion Models and Accuracy Evaluation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 10, pp. 965–980, 1992.
- [124] J. Heikkila, O. Silvén, and I. Oulu, “A Four-step Camera Calibration Procedure with Implicit Image Correction,” pp. 1106–1112, 1997.

APPENDIX A: SERVO MOTOR CONTROL SUB-SYSTEM OBJECT

Codes are also available at: https://github.com/amankhan47/Aman_SPM_TUNI.git

Program 1. Laser steering system

%% This program can be used for steering a laser beam to photorespon-
sive robot's parts detected by a machine vision system.
%A machine vision system serves the location information of the robot
legs and other features. Different control algorithm can be added to
the position_control() function.

```

classdef laser_steering_system<handle
    properties
        %tcp ip connection variable
        tcp_connection;
        recv_data_size = 8;
        data_receive;

        %servo controller
        fig_servo_ctrl;
        x_axis_controller=[];
        y_axis_controller=[];

        %robot legs; x,y:location; l:left leg; r:right leg
        xl,yl,x,y,xr,yr,
        rob_rot_ang; % rotation angle of the robot with the target
        k; %leg selection action
    end

    %% Initialize system
    methods
        function this = laser_steering_system1(this)
            %servo interface loading
            if isempty(this.fig_servo_ctrl) || ~is-
valid(this.fig_servo_ctrl)
                servo_interface_loading(this)
            end

            %connection to the machine vision server
            if isempty(this.tcp_connection)
                initiate_server_connection(this);
            end

            %iteration of steering operation
            i=1;
            while i<100
                %receive data from the machine vision server
                data_rec(this)
                %calibration and send comment to servo motor control-
ler
                position_control(this)
                i=i+1;
            end
        end
    end
end

```

```

        end
    end

    methods
        %% servo interface loading
        function servo_interface_loading(this)
            %% initialization of servo controller serial numbers
            serial_number_1 =27253885;
            serial_number_2 =27501118;

            waitbar_h=waitbar(.1,'Please wait, Loading ActiveX Con-
troller');

            f=.7; %magnification factor of the fig_servo_ctrl conten-
ier

            fpos(1) = 100;
            fpos(2) = 100;
            fpos(3) = f*480; % window width
            fpos(4) = f*2*300; % window height
            this.fig_servo_ctrl = figure('Position',
fpos,'Menu','None','Name','Controller Interface');

            % activeX controller
            this.x_axis_controller = actxcontrol('MGMOTOR.MGMo-
torCtrl.1',[f*10 f*295 f*450 f*300 ], this.fig_servo_ctrl);
            this.y_axis_controller = actxcontrol('MGMOTOR.MGMo-
torCtrl.1',[f*10 f*1 f*450 f*300 ], this.fig_servo_ctrl);

            % initialize and start control
            this.x_axis_controller.StartCtrl;
            this.y_axis_controller.StartCtrl;

            % set the serial number
            waitbar(.2,waitbar_h,'Setting the serial number')
            set(this.x_axis_controller,'HWSerialNum', serial_num-
ber_1);
            pause(1)
            set(this.y_axis_controller,'HWSerialNum', serial_num-
ber_2);

            % identify the device serial number
            waitbar(.5,waitbar_h,'Identifying the devices')
            this.x_axis_controller.Identify;
            pause(.5)
            this.y_axis_controller.Identify;
            % waiting for the GUI loading;
            pause(5);

            % Moving to home position
            waitbar(.7,waitbar_h,'Moving to home position')
            this.x_axis_controller.MoveHome(0,0);
            pause(1)
            this.y_axis_controller.MoveHome(0,0);
            waitbar(1,waitbar_h,'Done!')
            delete(waitbar_h);
        end
    end
end

```

```

%% vision server connection
function initiate_server_connection(this)
    % establish connection with machine vision server
    % Connection data
    host = '127.0.0.1';
    port = 4000;
    timeout = 60;

    % Configuration and connection
    this.tcp_connection=tcpip(host,port,'NetworkRole','serv-
er');

    this.tcp_connection.timeout = timeout;
    this.tcp_connection;
    % Wait for connection
    disp('Waiting for connection');
end

%% data receive
function data_rec(this)
    fopen(this.tcp_connection);
    fwrite(this.tcp_connection,'send');
    disp('data sent to vision soft')
    disp('Connection OK');

    this.data_receive =fread(this.tcp_conne-
tion,this.recv_data_size,'double');
    disp(this.data_receive)

    % check data integrity
    [row,~]=size(this.data_receive);
    if row==this.recv_data_size && (this.data_receive(8)==99)
        %leg positions: 1: left leg R:right leg
        this.xl=this.data_receive(1);
        this.yl=this.data_receive(2);

        this.x=this.data_receive(3);
        this.y=this.data_receive(4);

        this.xr=this.data_receive(5);
        this.yr=this.data_receive(6);

        %rotation angle of the robot
        this.rob_rot_ang=this.data_receive(7);

    end
    fclose(this.tcp_connection);
end
%% selection of leg manually
function set_k(this)
    val=(input('select an action for the robot: 1,2,3 \n>>'));
    this.k = val;
end
%% select the leg location (Different control algorithm can be
added)
function position_control(this)

    if this.k==1

```

```

        calibration_and_move_servo(this,this.xl,this.yl);
    elseif this.k==2
        calibration_and_move_servo(this,this.x,this.y);
    elseif this.k==3
        calibration_and_move_servo(this,this.xr,this.yr);
    end
end
%% steer laser beam to the desired location
function calibration_and_move_servo(this,x_move,y_move)
    %% calibration data
    xo = 7.40389;
    xf = 5.34398;
    x_in_mm = 10*9.305264;

    yo =9.42645;
    yf =6.99811;
    y_in_mm = 8*9.305264;

    x_axis = (xo-((xo-xf)*x_move)/x_in_mm)+.0534+.0029;
    y_axis = (yo-((yo-yf)*y_move)/y_in_mm)-.0670;

    %% move servo motor
    if x_axis>0
        this.x_axis_controller.SetAbsMovePos(0,x_axis);
        this.x_axis_controller.MoveAbsolute(0,1==0);
    end

    if y_axis>0
        this.y_axis_controller.SetAbsMovePos(0,y_axis);
        this.y_axis_controller.MoveAbsolute(0,1==1);
    end
    pause(.01)
end
end
end
end
end

```


APPENDIX B: VIDEO ANALYSIS TOOL FOR ANGLE MEASUREMENT (USER INTERFACE)

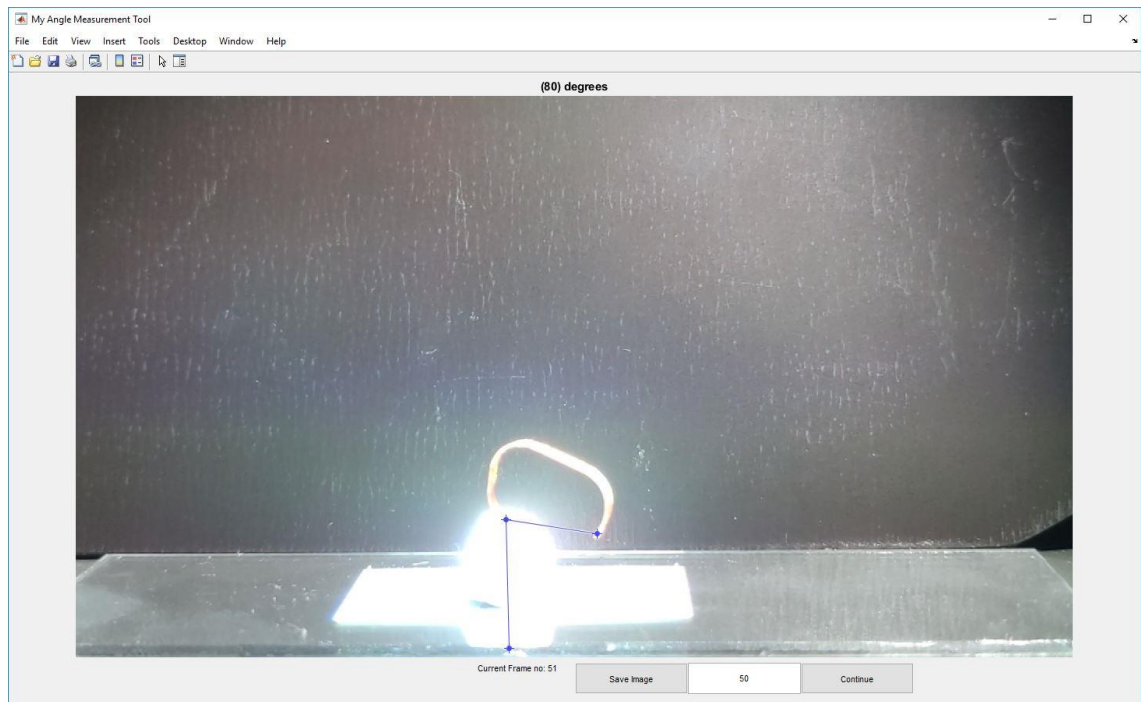


Figure (i): Measuring bending deformation of an LCN strip

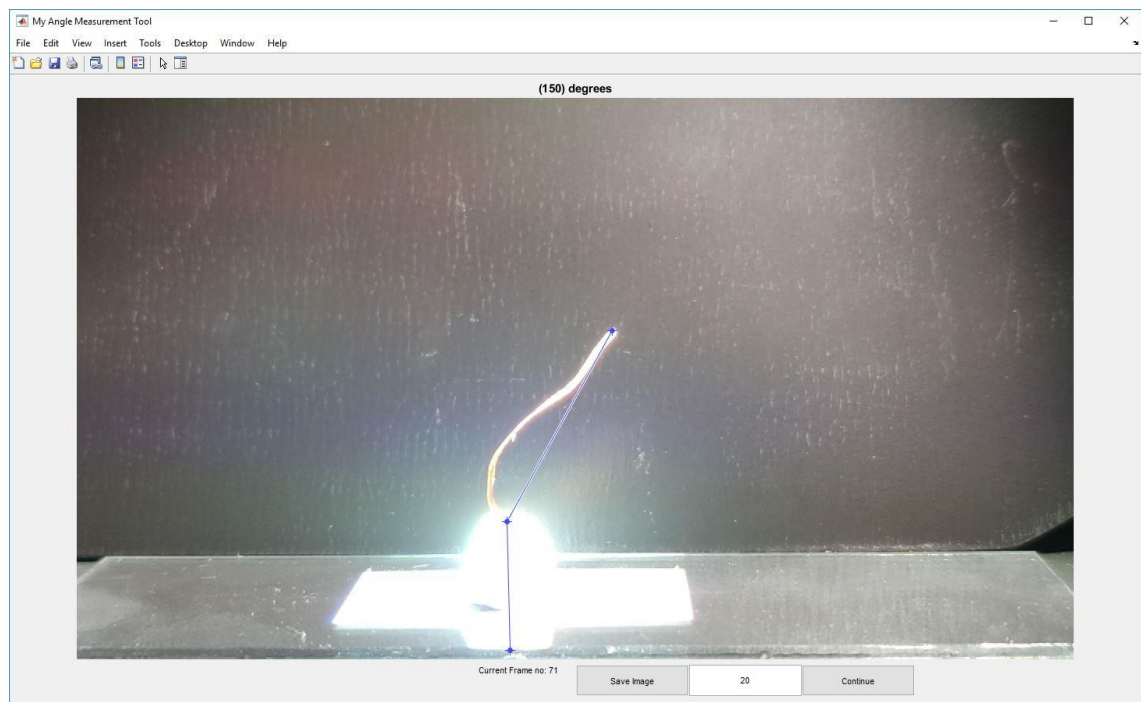


Figure (ii): Measuring bending deformation of an LCN strip

Code is available at: https://github.com/amankhan47/Aman_SPM_TUNI.git

APPENDIX C: DEVELOPED DEEP Q LEARNING PROGRAM

Codes are also available at: https://github.com/amankhan47/Aman_SPM_TUNI.git

Program 2. Environment

```

%% This program is used for creating RL environment and simulator

classdef robotenvironment< rl.env.MATLABEnvironment
    %% robot_environment: Defining environment for the photoresponsive
    robot.
    properties
        %x,y-axis location of the destination target
        xt=300;
        yt=300;

        %boundary circle
        Lx=500; %x limit of boundary circle
        Ly=500; %y limit of boundary circle
        xc; % center location,x
        yc; % center location,y
        r; %radius

        %robot properties
        k; %current robot leg
        rob_rot_ang=-45; %orientation of the robot body
        angle_btn_rob_trget; % angle between robot and target
        w; %robot width in the simulator
        h; %robot hight in the simulator
        V1=[]; %translation matrix of the robot body
        % current position of the robot
        x=150;
        y=200;
        % distance between robot and target
        d;
        pre_d=1; %previous distance

        % Reward
        Reward; %total reward for an action
        at_destination=false; %initializing flag for checking robot at
destination or not
        is_in_circle=true; %initializing flag for checking robot is
indide the circle or not
        penalty_not_in_circle= -100;% setting penalty for not in the
circle

        %simulator figure container handler
        handel;

        %plot robot trajectory; most useful for checking learned pol-
icity
        trace=true;
        %trace figure container handler
        ax;
    end
end

```

```

        %(optional)counter for changing target location at regular in-
    terval
        count=0;
    end

    properties
        %initializing observational states: [this.d;this.an-
    gle_btn_rob_trget;this.x;this.y;this.rob_rot_ang;this.xt;this.yt]
        State = zeros(7,1)
    end

    properties(Access = protected)
        %initializing flag for indicating episode ends
        IsDone = false
    end

%% required methods for reinforcement learning
    methods
        % creating environment instance
        function this = robotenvironment()
            %initializing observation states
            ObservationInfo = rlNumericSpec([7 1]);
            ObservationInfo.Name = 'Robot states';
            ObservationInfo.Description = 'd,an-
    gle_btn_rob_trget,x,y,rob_rot_ang';

            %initializing Action states settings
            ActionInfo = rlFiniteSetSpec([1 2 3]);
            ActionInfo.Name = 'Movement Action';

            % implementing reinforcement learning environment
            this = this@rl.env.MATLABEnvironment(ObservationInfo,Ac-
    tionInfo);

            % initializing property values
            updateActionInfo(this);
        end

        % simulating environment with one action
        function [Observation,Reward,IsDone,LoggedSignals] =
    step(this,Action)
            LoggedSignals = [];
            % initialize the robot body and boundary parameters
            robot(this);
            %set action
            this.k=Action;
            %emulate and calculate the effect of the action
            position_control(this,Action);
            %update observation states
            this.State = [this.d;this.an-
    gle_btn_rob_trget;this.x;this.y;this.rob_rot_ang;this.xt;this.yt];
            Observation = this.State;
            %check terminal conditions
            IsDone=(~this.is_in_circle)||this.at_destination;
            this.IsDone = IsDone;
            %calculate rewards
            Reward = getReward(this);
            %notify the change
            notifyEnvUpdated(this);
    end

```

```

end

%% set initial condition after every terminal episode
function InitialObservation = reset(this)
    this.k=2;
    Action=this.k;
    this.x=150;
    this.y=200;
    this.rob_rot_ang=-45;
    this.xt=150;
    this.yt=400;
    this.pre_d=this.d;
    robot(this);
    position_control(this,Action);
    InitialObservation = [this.d;this.angle_btn_rob_trget;this.x;this.y;this.rob_rot_ang;this.xt;this.yt];
    this.State = InitialObservation;
    notifyEnvUpdated(this);
end
end
%% methods related to action effect measurement and validate values
methods
    % robot and boundary parameter
    function robot(this)
        %robot body size
        this.w = this.Lx/40;
        this.h = this.Ly/60;
        % circle center and the radius from the center
        this.xc=this.Lx/2;
        this.yc=this.Ly/2;
        this.r=min(this.Lx/2,this.Ly/2);
    end

    %% emulate and calculate the effect of the action
    function position_control(this,Action)
        envUpdatedCallback(this)
        if ~ismember(Action,this.ActionInfo.Elements)
            error('Action must be 1,2,3 but %g is occurred.',Action);
        end
        this.k=Action;
        if this.k==1
            % set clockwise angle
            this.rob_rot_ang=this.rob_rot_ang-25;
            % calculate the effect of the action
            measurment(this);
        elseif this.k==2
            if this.rob_rot_ang<0
                % move forward
                this.x=this.x+20*(cos(deg2rad(this.rob_rot_ang+90)));
                this.y=this.y+20*(sin(deg2rad(this.rob_rot_ang+90)));
                measurment(this);
            else
                this.x=this.x-
                20*(cos(deg2rad(this.rob_rot_ang+180+90)));
                this.y=this.y-
                20*(sin(deg2rad(this.rob_rot_ang+180+90)));
                measurment(this);
            end
        elseif this.k==3

```

```

        % set anticlockwise angle
        this.rob_rot_ang=this.rob_rot_ang+25;
        measurment(this);
    end
end
%% calculate the effect of the action
function measurment(this)
    %distance between target and robot
    this.pre_d=this.d;
    this.d=sqrt(abs((this.xt-this.x)^2+(this.yt-this.y)^2));

    %rotation and translation matrix computation
    c = cos(deg2rad(this.rob_rot_ang));
    s = sin(deg2rad(this.rob_rot_ang));
    R = [c -s; s c]; %rotation matrix
    T = [R [this.x this.y]'; zeros(1,3)]; %translation matrix

    %robot structure matrix
    V0 = [ -this.w this.w this.w 0 -this.w ;
          -this.h -this.h this.h this.h*2.5 this.h ;
          ones(1,5) ];
    this.V1 = T*V0; %translation matrix of the robot body

    %creating vector to find angle
    x2=this.V1(1,4);
    y2=this.V1(2,4);
    v1 = [x2-this.x y2-this.y];
    v2 = [this.xt-this.x this.yt-this.y];

    % find angle between robot and target
    angle_btn_rob_trget_in_red =
    acos(dot(v1,v2)/(norm(v1)*norm(v2)));
    %conversion to degrees
    if isfinite(angle_btn_rob_trget_in_red)
        if det([v1;v2])<=0
            this.angle_btn_rob_trget = (an-
            gle_btn_rob_trget_in_red * (180/pi));
        else
            this.angle_btn_rob_trget =-(an-
            gle_btn_rob_trget_in_red * (180/pi));
        end
    else
        this.angle_btn_rob_trget=0;
    end

    %check the destination reached or not
    this.at_destination=(this.d<30);

    %checking current position inside the circle
    this.is_in_circle=((this.x-this.xc).^2+(this.y-
    this.yc).^2<=this.r^2);
end

%% update action states
function updateActionInfo(this)
    this.ActionInfo.Elements = [1 2 3];
end

%% Reward function
function Reward = getReward(this)

```

```

        if ~this.IsDone
            Reward = reward(this);
        elseif ~this.is_in_circle
            Reward = this.penalty_not_in_circle;
        else
            Reward = 100; %positive reward for achieving the tar-
get
        end
    end
end

function rwd = reward(this)

    if (this.d)<(this.pre_d)
        r1=5; %positive reward for going towards the target
    elseif (this.d)>(this.pre_d)
        r1=-2; %negative reward for going away from the target
    else
        r1=0;
    end

    % reward for maintaining low angle
    if abs(this.angle_btn_rob_trget)<25
        r2=1;
    else
        r2=-5;
    end

    % penalty for taking more step
    r3=-1;

    %total reward for the action
    rwd=r1+r2+r3;
    % disp(rwd)
end

%% visualization
function plot(this)
    envUpdatedCallback(this)
end

%% properties validation
function set.State(this,value)
    validateattributes(value,{'numeric'},{'fi-
nite','real','vector','numel',7},'', 'State');
    this.State = double(value(:));
    notifyEnvUpdated(this);
end
function set.k(this,value)
    validateattributes(value,{'numeric'},{'fi-
nite','real','positive','scalar'},'', 'k');
    this.k = value;
    notifyEnvUpdated(this);
end
function set.x(this,value)
    validateattributes(value,{'numeric'},{'fi-
nite','real','scalar'},'', 'x');
    this.x = value;
end
function set.y(this,value)

```

```

        validateattributes(value,{'numeric'},{'fi-
nite','real','scalar'},'', 'y');
        this.y = value;
    end
    function set.xt(this,value)
        validateattributes(value,{'numeric'},{'fi-
nite','real','positive','scalar'},'', 'xt');
        this.xt = value;
    end
    function set.yt(this,value)
        validateattributes(value,{'numeric'},{'fi-
nite','real','positive','scalar'},'', 'yt');
        this.yt = value;
    end
    function set.penalty_not_in_circle(this,value)
        validateattributes(value,{'numeric'},{'real','fi-
nite','scalar'},'', 'PenaltyorFalling');
        this.penalty_not_in_circle = value;
    end
end
%% simulator visualization
methods (Access = protected)
    %update visualization every time the environment is updated
    function envUpdatedCallback(this)
        simulator(this)
    end
end

%% simulator
function simulator(this)
    % create a simulator container if not available
    if isempty(this.handel) || ~isvalid(this.handel)
        this.handel = figure(...
            'ToolBar','none',...
            'NumberTitle','off',...
            'Name','LCN Robot Position Simulator and Visualiz-
er',...
            'Visible','on',...
            'MenuBar','none');
        figure(this.handel);
        f=clf;
        ha = gca(f);
        ha.XLim=[0 this.Lx];
        ha.YLim=[0 this.Ly];
        grid(ha, 'off');
        hold(ha, 'on');
    else
        figure(this.handel);
        f=clf;
        ha = gca(f);
        ha.XLim=[0 this.Lx];
        ha.YLim=[0 this.Ly];
        grid(ha, 'off');
        hold(ha, 'on');
    end

    %if robot body is not initiated before e.g. before taking
    %first step
    if isempty(this.V1)
        c = cos(deg2rad(this.rob_rot_ang));
        s = sin(deg2rad(this.rob_rot_ang));
        R = [c -s;s c]; %rotation matrix
    end
end

```

```

trix
        T = [R [this.x this.y]';zeros(1,3)]; %translation ma-
robot(this);
%robot structure matrix
V0 = [ -this.w      this.w      this.w      0
      -this.h      -this.h      this.h      this.h*2.5
      ones(1,5)    ];
this.V1 = T*V0; %translation matrix of the robot body
end

%robot-body location
vx = this.V1(1,1:5);
vy = this.V1(2,1:5);

%find the body, boundary circle and target point
body = findobj(ha, 'Tag', 'body');
boundary_circle = findobj(ha, 'Tag', 'boundary_circle');
target_point = findobj(ha, 'Tag', 'target_point');

if isempty(body)
    patch(vx,vy,[0.3010 0.7450 0.9330], 'Tag', 'body');
else
    body.XData = vx;
    body.YData = vy;
end

if isempty(boundary_circle)
    this.xc=this.Lx/2;
    this.yc=this.Ly/2;
    this.r=min(this.Lx/2,this.Ly/2);
    boundary_circle=viscircles([this.xc,this.yc],this.r);
    boundary_circle.Tag='boundary_circle';
end

if isempty(target_point)
    target_point= viscircles([this.xt,this.yt],this.h/5);
    target_point.Tag='target_point';
end

% plot robot trajectory; most useful for checking learned
policy
if this.trace==true
    if isempty(this.ax) || ~isvalid(this.ax)
        figure();
        this.ax = axes;
    else
        copyobj(findobj(ha, 'Tag', 'body'),this.ax);
        copyobj(findobj(ha, 'Tag', 'target_point'),this.ax);
        copyobj(findobj(ha, 'Tag', 'boundary_cir-
cle'),this.ax);
        hold(this.ax, 'on')
    end
end
drawnow();
hold(ha, 'off')
end
end
end
end

```


Program 3. DQN and agent training

```

%% This program is used for creating DQN and agent training

%% creating environment instance
rob_env = robotenvironment;
%validate Environment
validateEnvironment(rob_env)

%random number seed
rng(0);

% deep neural network
state_path = [
    imageInputLayer([7 1 1], 'Normalization', 'none', 'Name', 'observational_state')
    fullyConnectedLayer(12, 'Name', 'Critic_State_FC1')
    reluLayer('Name', 'Critic_Relul')
    fullyConnectedLayer(12, 'Name', 'Critic_State_FC2')];

action_path = [
    imageInputLayer([1 1 1], 'Normalization', 'none', 'Name', 'action_state')
    fullyConnectedLayer(12, 'Name', 'Critic_Action_FC1')];

common_path = [
    additionLayer(2, 'Name', 'addition')
    reluLayer('Name', 'Critic_Common_Relul')
    fullyConnectedLayer(1, 'Name', 'output')];

critic_network = layerGraph(state_path);
critic_network = addLayers(critic_network, action_path);
critic_network = addLayers(critic_network, common_path);
critic_network = connectLayers(critic_network, 'Critic_State_FC2', 'addition/in1');
critic_network = connectLayers(critic_network, 'Critic_Action_FC1', 'addition/in2');

%plot network
figure
plot(critic_network)

% critic parameters
critic_opts = rlRepresentationOptions('LearnRate',0.01,'GradientThreshold',1);

% get observation and action states info
obs_info = getObservationInfo(rob_env);
act_info = getActionInfo(rob_env);

% setting RL agent
critic = rlRepresentation(critic_network,obs_info,act_info,'Observation',{ 'observational_state' }, 'Action',{ 'action_state' },critic_opts);

% setting up DQN agent parameters
agent_Opts = rlDQNAgentOptions(...
    'UseDoubleDQN',false, ...
    'TargetUpdateMethod',"periodic", ...
    'TargetUpdateFrequency',4, ...

```

```

    'NumStepsToLookAhead',1,...
    'ExperienceBufferLength',10000, ...
    'DiscountFactor',0.9, ...
    'MiniBatchSize',32,...
    'SampleTime',.00001,...
    'ResetExperienceBufferBeforeTraining',0,...
    'SaveExperienceBufferWithAgent',1);

%create DQL agent
agent = rlDQNAgent(critic,agent_Opts);

% tanning parameters
train_Opts = rlTrainingOptions(...
    'MaxEpisodes', 3000, ...
    'MaxStepsPerEpisode', 20, ...
    'Verbose', false, ...
    'Plots','training-progress',...
    'StopTrainingCriteria','AverageReward',...
    'StopTrainingValue',300,...
    'SaveAgentCriteria',"EpisodeReward",...
    'SaveAgentValue', 100);

%% initiate tanning
%use previously saved agent
%load('savedAgents/ran_ang_1_3.mat','agent');
training = train(agent,rob_env,train_Opts);

% save the trained agent
save(train_Opts.SaveAgentDirectory + "/diff_tar_ran_all_ac-
tion.mat",'agent')

```

Program 4. Run_Simulaton

```
%% This program is used for evaluating the learned polices

%% creating environment instance
rob_env = robotenvironment;
%validate Environment
validateEnvironment(rob_env)

% set target location
rob_env.xt=250;
rob_env.yt=450;

%load trained agent
load('savedAgents/diff_tar_ran_all_action.mat','agent');

%set simulation option
sim_options = rlSimulationOptions('MaxSteps',50, 'NumSimulations',1);
experience = sim(rob_env,agent,sim_options);
totalReward = sum(experience.Reward)
```

APPENDIX D: DEVELOPED DEEP Q LEARNING PROGRAM FOR IMPLEMENTATION WITH MV AND LASER STEERING SYSTEM

Program 5. Environment

```

%% This program is used for creating RL environment implementing with
mv and laser steering system
classdef robotenvironment_MV< rl.env.MATLABEnvironment
    %% robotenvironment_MV: Defining environment for the photorespon-
sive robot.

    %% Properties (set properties' attributes accordingly)
    properties
        %x,y-axis location of the destination target
        xt=300;
        yt=300;

        %boundary circle
        Lx=500; %x limit of boundary circle
        Ly=500; %y limit of boundary circle
        xc; % center location,x
        yc; % center location,y
        r; %radius

        %robot properties
        k; %current robot leg
        rob_rot_ang=-45; %orientation of the robot body
        angle_btn_rob_trget; % angle between robot and target
        proximity_to_target=30; % minimum distance between robot and
target
        w; %robot width in the simulator
        h; %robot height in the simulator
        V1=[]; %translation matrix of the robot body
        % current position and robot legs: x,y:location; l:left leg;
r:right leg
        xr;
        yr;
        x;
        y;
        xl;
        yl;
        % distance between robot and target
        d;
        pre_d=1; %previous distance

        % Reward
        Reward; %total reward for a action
        at_destination=false; %initializing flag for checking robot at
destination or not
        is_in_circle=true; %initializing flag for checking robot is
inside the circle or not
        penalty_not_in_circle= -100;% setting penalty for not in the
circle

```

```

%simulator figure container handler
handel;

%plot robot trajectory; most useful for checking learned pol-
icy
trace=true;
%trace figure container handler
ax;

%tcp ip connection variable
tcp_connection;
recv_data_size = 8;
data_receive;

%servo controller
fig_servo_ctrl;
x_axis_controller=[];
y_axis_controller=[];
end

properties
%initializing observational states: [this.d;this.an-
gle_btn_rob_trget;this.x;this.y;this.rob_rot_ang;this.xt;this.yt]
State = zeros(7,1)
end

properties(Access = protected)
%initializing flag for indicating episode ends
IsDone = false
end

%% required methods for reinforcement learning
methods
%% creating environment instance
function this = robotenvironment_MV()
%initializing observation states
ObservationInfo = rlNumericSpec([7 1]);
ObservationInfo.Name = 'Robot states';
ObservationInfo.Description = 'd,an-
gle_btn_rob_trget,x,y,rob_rot_ang';

%initializing Action states settings
ActionInfo = rlFiniteSetSpec([1 2 3]);
ActionInfo.Name = 'Movement Action';

% implementing reinforcement learning environment
this = this@rl.env.MATLABEnvironment(ObservationInfo,Ac-
tionInfo);

% initializing property values
updateActionInfo(this);
end

%% environment with one action
function [Observation,Reward,IsDone,LoggedSignals] =
step(this,Action)
LoggedSignals = [];
% initialize the robot body and boundary parameters for
% visualizer

```

```

robot(this);
%set action
this.k=Action;
%select the lage location which is retrieved from MV
position_control(this,Action);

% update observation states
this.State = [this.d;this.an-
gle_btn_rob_trget;this.x;this.y;this.rob_rot_ang;this.xt;this.yt];
Observation = this.State;

% check terminal condition
IsDone=(~this.is_in_circle)||this.at_destination;
this.IsDone = IsDone;

%%calculate rewards
Reward = getReward(this);
%notify the change
notifyEnvUpdated(this);
end

%% set initial condition after every terminal episode
function InitialObservation = reset(this)

%connection to the machine vision server
if isempty(this.t)
    initiate_server_connection(this)
end

if isempty(this.fig_servo_ctrl) || ~is-
valid(this.fig_servo_ctrl)
    servo_interface_loading(this)
end

%data receive for MV system
data_rec(this)
%default action
this.k=2;
Action=this.k;
this.pre_d=this.d;
robot(this);
%select the large location which is retrieved from MV
position_control(this,Action);
InitialObservation = [this.d;this.an-
gle_btn_rob_trget;this.x;this.y;this.rob_rot_ang;this.xt;this.yt];
this.State = InitialObservation;

% (optional) use notifyEnvUpdated to signal that the
% environment has been updated (e.g. to update visualiza-
tion)
notifyEnvUpdated(this);
this.count=this.count+1;
end
end
%% Optional Methods (set methods' attributes accordingly)
methods
%% robot and boundary parameter
function robot(this)
    this.w = this.Lx/40;

```

```

        this.h = this.Ly/60;
        this.xc=this.Lx/2;
        this.yc=this.Ly/2;
        this.r=min(this.Lx/2,this.Ly/2);
    end

    %% emulate and calculate the effect of the action
    function position_control(this,Action)
        if ~ismember(Action,this.ActionInfo.Elements)
            error('Action must be 1,2,3 but %g is oc-coured.',Ac-
tion);
        end
        this.k=Action;
        if this.k==1
            calibration_and_move_servo(this,this.xl,this.yl);
        elseif this.k==2
            calibration_and_move_servo(this,this.x,this.y);
        elseif this.k==3
            calibration_and_move_servo(this,this.xr,this.yr);
        end
        data_rec(this);
        measurment(this);

    end

    %% calculate the the effect of the action
    function measurment(this)
        %%distance between target and robot
        this.pre_d=this.d;
        this.d=sqrt(abs((this.xt-this.x)^2+(this.yt-this.y)^2));

        %%creating vector to find angle
        x2=(this.xl+this.xr)/2;
        y2=(this.yl+this.yr)/2;
        v1 = [x2-this.x y2-this.y];
        v2 = [this.xt-this.x this.yt-this.y];

        %% find angle between robot and target
        angle_btn_rob_trget_in_red =
acos(dot(v1,v2)/(norm(v1)*norm(v2)));
        %%conversion to degrees
        if isfinite(angle_btn_rob_trget_in_red)
            if det([v1;v2])<=0
                this.angle_btn_rob_trget = (an-
gle_btn_rob_trget_in_red * (180/pi));
            else
                this.angle_btn_rob_trget =-(an-
gle_btn_rob_trget_in_red * (180/pi));
            end
        else
            this.angle_btn_rob_trget=0;
        end

        %%check the destination reached or not
        this.at_destination=(this.d<this.proximity_to_target);

        %%checking robot current position inside the circle
        this.is_in_circle=((this.x-this.xc).^2+(this.y-
this.yc).^2<=this.r^2);
    end
end

```

```

%% update action states
function updateActionInfo(this)
    this.ActionInfo.Elements = [1 2 3];
end

%% Reward function
function Reward = getReward(this)
    if ~this.IsDone
        Reward = reward(this);
    elseif ~this.is_in_circle
        Reward = this.penalty_not_in_circle;
    else
        Reward = 100; %positive reward for achieving the tar-
get
    end
end

function rwd = reward(this)

    if (this.d)<(this.pre_d)
        r1=5; %positive reward for going towards the target
    elseif (this.d)>(this.pre_d)
        r1=-2; %negative reward for going away from the target
    else
        r1=0;
    end

    % reward for maintaining low angle
    if abs(this.angle_btn_rob_trget)<25
        r2=1;
    else
        r2=-5;
    end

    % penalty for taking more step
    r3=-1;

    %total reward for the action
    rwd=r1+r2+r3;
    disp(rwd)

end
%% vision server connection
function initiate_server_connection(this)
    % establish connection with machine vision server
    % Connection data
    host = '127.0.0.1';
    port = 4000;
    timeout = 60;

    % Configuration and connection
    this.tcp_connection=tcpip(host,port,'NetworkRole','serv-
er');

    this.tcp_connection.timeout = timeout;
    this.tcp_connection;
    % Wait for connection
    disp('Waiting for connection');

```



```

end

%% data receive
function data_rec(this)
    fopen(this.tcp_connection);
    fwrite(this.tcp_connection,'send');
    disp('data sent to vision soft')
    disp('Connection OK');

    this.data_receive =fread(this.tcp_connection,this.recv_data_size,'double');
    disp(this.data_receive)

    % check data integrity
    [row,~]=size(this.data_receive);
    if row==this.recv_data_size && (this.data_receive(8)==99)
        %leg positions: l:left leg R:right leg
        this.xl=this.data_receive(1);
        this.yl=this.data_receive(2);

        this.x=this.data_receive(3);
        this.y=this.data_receive(4);

        this.xr=this.data_receive(5);
        this.yr=this.data_receive(6);

        %rotation angle of the robot
        this.rob_rot_ang=this.data_receive(7);

    end
    fclose(this.tcp_connection);
end

%% servo interface loading
function servo_interface_loading(this)
    %initialization of servo controller serial numbers
    serial_number_1 =27253885;
    serial_number_2 =27501118;

    waitbar_h=waitbar(.1,'Please wait, Loading ActiveX Controller');

    f=.7; %magnification factor of the fig_servo_ctrl container
    fpos(1) = 100;
    fpos(2) = 100;
    fpos(3) = f*480; % window width
    fpos(4) = f*2*300; % window height
    this.fig_servo_ctrl = figure('Position',
fpos,'Menu','None','Name','Controller Interface');

    % activeX controller
    this.x_axis_controller = actxcontrol('MG MOTOR.MGMotorCtrl.1',[f*10 f*295 f*450 f*300 ], this.fig_servo_ctrl);
    this.y_axis_controller = actxcontrol('MG MOTOR.MGMotorCtrl.1',[f*10 f*1 f*450 f*300 ], this.fig_servo_ctrl);

```

```

% initialize and start control
this.x_axis_controller.StartCtrl;
this.y_axis_controller.StartCtrl;

% set the serial number
waitbar(.2,waitbar_h,'Setting the serial number')
set(this.x_axis_controller,'HWSerialNum', serial_num-
ber_1);
pause(1)
set(this.y_axis_controller,'HWSerialNum', serial_num-
ber_2);

% identify the device serial number
waitbar(.5,waitbar_h,'Identifying the devices')
this.x_axis_controller.Identify;
pause(.5)
this.y_axis_controller.Identify;
% waiting for the GUI loading;
pause(5);

% Moving to home position
waitbar(.7,waitbar_h,'Moving to home position')
this.x_axis_controller.MoveHome(0,0);
pause(1)
this.y_axis_controller.MoveHome(0,0);
waitbar(1,waitbar_h,'Done!')
delete(waitbar_h);
end

%%steer laser beam to the desired location
function calibration_and_move_servo(this,x_move,y_move)
    %% calibration data
    xo = 7.40389;
    xf = 5.34398;
    x_in_mm = 10*9.305264;

    yo =9.42645;
    yf =6.99811;
    y_in_mm = 8*9.305264;

    x_axis = (xo-((xo-xf)*x_move)/x_in_mm)+.0534+.0029;
    y_axis = (yo-((yo-yf)*y_move)/y_in_mm)-.0670;

    %% move servo motor
    if x_axis>0
        this.x_axis_controller.SetAbsMovePos(0,x_axis);
        this.x_axis_controller.MoveAbsolute(0,1==0);
    end

    if y_axis>0
        this.y_axis_controller.SetAbsMovePos(0,y_axis);
        this.y_axis_controller.MoveAbsolute(0,1==1);
    end
    pause(.01)
end

```

```

%% visualization
function plot(this)
    envUpdatedCallback(this)
end

%% properties validation
function set.State(this,value)
    validateattributes(value,{'numeric'},{'finite','real','vector','numel',7},'', 'State');
    this.State = double(value(:));
    notifyEnvUpdated(this);
end
function set.k(this,value)
    validateattributes(value,{'numeric'},{'finite','real','positive','scalar'},'', 'k');
    this.k = value;
    notifyEnvUpdated(this);
end
function set.x(this,value)
    validateattributes(value,{'numeric'},{'finite','real','scalar'},'', 'x');
    this.x = value;
end
function set.y(this,value)
    validateattributes(value,{'numeric'},{'finite','real','scalar'},'', 'y');
    this.y = value;
end
function set.xt(this,value)
    validateattributes(value,{'numeric'},{'finite','real','positive','scalar'},'', 'xt');
    this.xt = value;
end
function set.yt(this,value)
    validateattributes(value,{'numeric'},{'finite','real','positive','scalar'},'', 'yt');
    this.yt = value;
end
function set.penalty_not_in_circle(this,value)
    validateattributes(value,{'numeric'},{'real','finite','scalar'},'', 'PenaltyorFalling');
    this.penalty_not_in_circle = value;
end
end
%% simulator visualization
methods (Access = protected)
%update visualization every time the environment is updated
function envUpdatedCallback(this)
    simulator(this)
end

%% simulator
function simulator(this)
    % create a simulator container if not available
    if isempty(this.handel) || ~isvalid(this.handel)
        this.handel = figure(...
            'ToolBar','none',...
            'NumberTitle','off',...
            'Name','LCN Robot Position Simulator and Visualizer',...
er',...

```

```

        'Visible','on',...
        'MenuBar','none');
figure(this.handel);
f=clf;
ha = gca(f);
ha.XLim=[0 this.Lx];
ha.YLim=[0 this.Ly];
grid(ha,'off');
hold(ha,'on');
else
figure(this.handel);
f=clf;
ha = gca(f);
ha.XLim=[0 this.Lx];
ha.YLim=[0 this.Ly];
grid(ha,'off');
hold(ha,'on');
end

%if robot body is not initiated before e.g. before taking
%first step
if isempty(this.V1)
c = cos(deg2rad(this.rob_rot_ang));
s = sin(deg2rad(this.rob_rot_ang));
R = [c -s;s c]; %rotation matrix
T = [R [this.x this.y]';zeros(1,3)]; %translation ma-
trix

robot(this);
%robot structure matrix
V0 = [ -this.w      this.w      this.w      0
      -this.h      -this.h      this.h      this.h*2.5
      ones(1,5)    ];
this.V1 = T*V0; %translation matrix of the robot body
end

%robot-body location
vx = this.V1(1,1:5);
vy = this.V1(2,1:5);

%find the body, boundary circle and target point
body = findobj(ha,'Tag','body');
boundary_circle = findobj(ha,'Tag','boundary_circle');
target_point = findobj(ha,'Tag','target_point');

if isempty(body)
patch(vx,vy,[0.3010 0.7450 0.9330],'Tag','body');
else
body.XData = vx;
body.YData = vy;
end

if isempty(boundary_circle)
this.xc=this.Lx/2;
this.yc=this.Ly/2;
this.r=min(this.Lx/2,this.Ly/2);
boundary_circle=viscircles([this.xc,this.yc],this.r);
boundary_circle.Tag='boundary_circle';
end
end

```

```

if isempty(target_point)
    target_point= viscircles([this.xt,this.yt],this.h/5);
    target_point.Tag='target_point';
end

%% robot trajectory; most useful for checking learned policy
icy
if this.trace==true
    if isempty(this.ax) || ~isvalid(this.ax)
        figure();
        this.ax = axes;
    else
        copyobj(findobj(ha,'Tag','body'),this.ax);
        copyobj(findobj(ha,'Tag','target_point'),this.ax);
        copyobj(findobj(ha,'Tag','boundary_circ-
cle'),this.ax);
        hold(this.ax,'on')
    end
end
drawnow();
hold(ha,'off')
end
end
end
end

```