UiO : Department of Informatics
University of Oslo

Design of a wall-climbing robot system

KlimBot - a legged robot for climbing bolted walls

Kim Stephen Bovim

Master's Thesis Autumn 2014

Design of a wall-climbing robot system

Kim Stephen Bovim

3rd November 2014

Abstract

The progress of computer science and mechanical and electrical engineering in the field of robotics has increased the applicability of robots for replacing human resources in the performance of repetitive and dangerous tasks. A lot of today's work on robots for oil platforms, manufacturing plants and other facilities providing potentially hazardous environments is based on adjusting the traditional industrial robots to operate in fixed coordinate systems. There is a need for enabling robots to move freely on walls and similar structures.

One of the major challenges of wall-climbing robots is attaining a secure and reliable grip to the wall. The object of this master's thesis has been to determine the feasibility and applicability of a wall-climbing robot that adheres to the wall by gripping on to bolts, with a control system enabling an operator to easily lead the robot over a considerable distance.

In order to do so, KlimBot, a wall-climbing prototype for bolted walls, has been designed, built, programmed and tested. KlimBot has been able to successfully climb horizontally and vertically on designated test walls. Additionally, a control system was implemented; enabling path generation and traversal over several bolts.

The test results achieved by KlimBot suggests that there is a great potential for achieving a highly reliable adhesion with the proposed approach. However, if such a wall-climbing robot was to be commercialized, the assumptions made in order for KlimBot to achieve satisfactory functionality would not hold. After discussing options for optimizing KlimBot's design and implementation, the thesis is concluded with a discussion concerning the tough challenges that have to be conquered in order for commercialization to be feasible.

Acknowledgement

To my beloved Silje Sæby Dybvik, Aslan and my closest family.
I am deeply grateful for all your love and support;
I would not have managed without you.

Contents

List of Figures

Part I

Introduction

Chapter 1

Introduction

1.1 Introduction

Technological progress and greater demands for security and life quality in today's society has, in general, increased the demand for automation. Wall-climbing robots have the prospect of being of great help in this automation by performing a variety of tasks that might be difficult, dangerous, time consuming and costly if performed by human beings. There has been done a fair amount of research in the field of wall-climbing robots, but the commercial use is still very limited. One of the biggest current challenges is to create a robot that can reliably stick to the wall. There has been proposed several different designs of wall-climbing robots with different approaches for adhesion and locomotion for meeting the wide variety of demands that exist.

During the work of this master's thesis, a legged wall-climbing robot prototype has been designed, built and programmed, that meet the challenge of sticking to the wall by gripping onto bolts on a designated wall. The robot has been named KlimBot. The name is a combination of "climbing robot" and the authors name, Kim Bovim. A control system has also been created, with the objective of enabling an operator to easily control the robot.

1.2 Master's thesis main problem

The objective of this master's thesis is to acquire the sufficient understanding and practical experience required for answering the following question.

Can there be made a reliable wall-climbing robot that grips to bolts on a designated wall, with a control system for path planning so that an operator easily can program the robots movements for the traversal of possibly hundreds of bolts, without having to physically direct each leg to each bolt, but simply tell the robot where to climb and what actions to perform?

1.3 Motivation

A lot of today's work on robots for oil platforms, manufacturing plants and other potentially hazardous facilities is based on adjusting the traditional industrial robots to operate in fixed coordinate systems. There is a need for enabling robots to move freely on walls and similar structures.

There has not been done much research on the feasibility and applicability of such robots, most likely because of the strict prerequisites for a designated wall. Nevertheless, the approach of wall-climbing robots for bolted walls is very interesting, because of the great underlying potential for acquiring a highly reliable adhesion. A secure grip to the wall could also facilitate the operation of heavy tools, which could lead to the ability of performing tasks that no current climbing robots are capable of.

The objective of the research of this master's thesis is to attain a better understanding of the how realistic and applicable such an approach is.

1.4 Summary

This master's thesis consists of three separate parts.

- *Introduction.* The first part contains two chapters, firstly, an introductory chapter, stating the master's thesis main problem as well as the objective and motivation.

  Secondly, a background chapter, describing previous approaches and the programs and tools used during the work of this thesis.

- *The project.* The second part consists of three chapters describing the research that has been done during during the work with KlimBot. The design phase, test process and the implementation of the KlimBot Control System are all described in detail.

- *Discussion.* The last part of this thesis consists of a discussion, firstly, the work with KlimBot is thoroughly discussed, before a discussion on the transition challenges from a test prototype to applicable wall-climbing robot follows.

  Finally, a conclusive chapter summarizes the research findings and the discussion that followed.

Chapter 2

Background

2.1 Robotics

Robotics is the field of engineering and science concerning robots, including the design, building and operation of these[33]. Robots are mechanical devices usually designed for acting as a tool for easing the operation of certain tasks, or replacing human beings in dangerous, repetitive or heavy labour. A certain degree of autonomy is often present. The idea of autonomy originates from ancient civilizations[27], but the first digital and programmable industrial robot, like we know them, was not created until the 1950s by George Devol[28], indicating that the field of robotics is not that old. Nowadays, commercial and industrial robots are used in numerous different domains for a wide range of tasks, such as manufacturing, surgery, weaponry, research and even space exploration. All robots have in common that they have some sort of mechanical construction, some electronic components and some programmed logic for operation.

Rigid industrial robot parts are referred to as *links*, which are connected by *joints* for facilitating relative movement between the links. Joints are typically said to be *revolute* when they are rotary like a hinge, or *prismatic* when they facilitate linear movement[45]. For revolute joints, the joint variables are named joint angles. However, since the joint values are represented as positional increments, not angles, in the rotational actuators used in this thesis, the joint variables have been referred to as positions throughout the text, for distinction. Similarly, the robot links have been given names describing their function.

2.2 Wall-climbing robots

Wall-climbing robots are imagined to aid in a broad variety of different tasks. The robots can be of help in accessing areas that might be difficult for humans to reach, such as the walls of tall buildings or instalments. These tall heights also pose a serious threat of human workers falling down. Additionally, robots have the potential to handle other hazardous environments that are dangerous to human beings.

Nuclear storage tanks present very dangerous conditions, with a risk of radioactive leakage. Radiation-hardened electronics can survive much higher levels of radiation than humans do. These radiation-hardened electronics have already been used for decades in space instalments to deal with the hostile environment and the effects of radiation trapped in the earth's magnetic field[5].

Robots can also be modified to tolerate extreme temperatures or rough weather conditions, both which are often present at oil rigs. Work on oil rigs represents a risky environment, and human manpower at oil rigs is an expensive resource. Fully functional robots can prove to be cost effective, as they are thought to perform tasks more efficient and reliable than human labour, doing fewer mistakes. Robots can be more effective both in time used on a specified task as well as time saved on not having to set up scaffolds and the likes, facilitating human labour. This will furthermore reduce the spells of halts in production, resulting in an additional increase of productivity and cost effectiveness.

Summed up, wall-climbing robots can be used for several different tasks that might be difficult to reach, dangerous, time consuming and costly.

The following is a few examples of tasks thought to be suitable for wall-climbing robots:

- *Welding seam inspection of nuclear storage tanks*[53]. The extremely dangerous conditions associated with nuclear storage tanks, with a constant risk of lethal leakage, present an excessively hazardous environment for a human workforce.

- *Spacecraft hull inspection and repair*[25]. The limited accessibility combined with a challenging and dangerous atmosphere complicate the performance of such tasks.

- *Surveillance*[34]. A climbing robot could be suitable for reaching tall lookouts in order to acquire a desirable overview, for reconnaissance, for gaining access to hard-to-reach areas, or for maintaining unnoticed during espionage investigations[52].

- *Inspection during shipbuilding and construction*. A robot has been designed that uses a colour camera for inspecting for rust and defects in structure or paint. It also uses the cameras for inspecting the geometric features of screws and bolts to check if they are at their required torque[4].

- *Cleaning*. In fact, there is a window cleaning robot installed here at the Department of Informatics at the University of Oslo. Rails are set up for conveyance around the roof of the building, and wires are used for hoisting the robot to the appropriate height.

- Other suitable tasks are maintenance of boiler tubing, performing preventive measures or rehabilitating by spray painting or sandblasting for anti-corrosion and anti-rust, as well as vacuum-blasting and lacquer coat thickness measurements[53].

2.3 Previous approaches

The research field of wall-climbing robots is still relatively young and is undergoing immense development. Several different wall-climbing robots have been made, with a great diversity with respect to size, design, applicability and behaviour. A perfect climbing robot would have to evaluated in the context of the environment and situations the robot is intended for. The broad spectre of possible application areas and desired behaviour has given rise to wide variety of suggested approaches, with different combinations of adhesion methods and means of locomotion applied. Some of the previous approaches as well as their advantages and disadvantages are described in the following text.

2.3.1 Adhesion

One major challenge for a wall-climbing robot is to attain an adhesion with sufficient reliability, and the different adhesion methods remains a main focus for research in the field of wall-climbing robots.

- *Negative pressure/vacuum suction.* The robot sticks to the wall by creating negative pressure between the wall and suction cups on the robot. These adhesion methods are light, but require a very smooth surface. Even slight cracks or obstacles could result in the robot losing its footing and consequently falling of the wall. . Furthermore, these methods are also not suitable for outer space projects (zero air pressure)[8][7].
- *Magnetic adhesion.* This is a highly reliable form for adhesion, but it's only applicable on ferromagnetic surfaces. It also requires a very heavy load of magnets, which reduces speed and increases power consumption[53].
- *Bionic suction.* Biologically inspired suction types, such as legs covered with dry micro fibres inspired by the toe hairs of the gecko. The thin hairs help the gecko stick to the wall by van der Waals forces, which are considered to be relatively weak. It might be sufficient for light robots, but is not suitable for tasks that require a heavy payload. There are also issues on how to keep the hairs clean, in order to keep the hairs sticky[20].
- *Hot melt adhesion.* Adhesion is acquired by using material with special thermal properties. Heat is applied in order to melt and soften the material, and adhesion forces are created by cooling the material until it returns to solid form. The adhesion forces created are relatively strong, but locomotion velocity is a challenge[31].
- *Gripping to the surface.* Some robots have used grippers with several fishing hooks as claws that allow them to grip onto uneven surfaces, such as brick walls. This could be a good idea for light payloads, for instance a camera for surveillance[43].

- *Tracks/rails*. Robots that follow tracks have a secure adhesion to the wall and can potentially carry heavy payloads. The downside is that they require a specialized and predefined track. The requirement of robot to follow these tracks, makes it less flexible in the sense of movement[30].

2.3.2 Locomotion

Another of the most considerable challenges of climbing on vertical walls, is related to locomotion, or rather the combination of adhesion and locomotion, the need for maintaining a secure grip whilst moving at the same time. Different types of locomotion methods for dealing with this challenge have been proposed. The three most common types of locomotion are the crawler, the wheeled and the legged type[42].

- *Crawlers*. Robots that crawl are potentially fast, and have the advantage of stable navigation on uneven terrain, but might not handle obstacles too well[26].

- *Wheeled*. The wheeled type robots can often achieve high velocities, but face some of the same issues as the crawler type with respect to passing obstacles or rough surfaces[41].

- *Legged*. Legged robots handle obstacles and cracks in the surface better, but are typically slower. These robots often have from two up to eight legs. The more limbs, the more stable adhesion and potential for carrying heavy payloads, but with a penalty of increased size and weight as well as requiring more complex control systems.

Additional types of locomotion are the tracked and cable-driven types, or combinations of any of the types mentioned[11].

2.4 Robot control systems

There is a wide variety of wall-climbing robots that might be operator-driven, semi-autonomous or autonomous. Most wall-climbing robots exert some degree of autonomy, either by planning and mapping their path and actions in real-time with the help of different sensors, or carrying out and repeating different preprogrammed tasks and operations. Wall-climbing robots that can be driven by an operator are usually controlled through wireless transmitters, with the help of some Human-Machine-Interface[13]. Remote control is especially desired when the robot operates in hard-to-reach locations or in very hazardous environments, letting the operator control the robot from a safe distance.

2.4.1 Programming methods

The most basic methods for programming industrial robots are the teach method, the lead through method and off-line programming. An additional possible approach is to use evolutionary computation for finding the appropriate way of moving. These methods are described in more detail below.

- *Teach method.* The teach method is carried out by using some sort of specialized teach pendant; typically a hand-held control device, often with some sort of graphical interface. With the teach pendant, you can control the robot, usually making small steps by altering joint angles and positions or by navigating in global or designated three dimensional Cartesian coordinate systems, saving all positions along the way. There is also the possibility off saving series of positions and reusing them in a programming sequence, saving time when repeatedly returning to the same positions[6].

- *The lead through method.* This method, which is occasionally referred to as the lead-by-the-nose method, is also performed by saving a collection of positions and small movements before putting them all together. The difference is that the initial movement of the robot is carried out by physically moving the robot from position to position, not aided by a controller or joystick[32].

- *Off-line programming.* Off-line programming is performed with software simulation programs, like ABB's RobotStudio[1], that uses CAD (Computer-Aided Design) models to generate position sequences. This method has the advantage of reduced down time, as the robot that is to be reprogrammed can continue its current operation during the stages of simulation, before the completed and updated instructional software is loaded to the robot[22].

- *Evolutionary Computation.* Evolutionary computational techniques, such as genetic or evolutionary algorithms, utilizes Darwinian principles and replicates biological evolutionary mechanisms such as reproduction, recombination, mutations and natural selection for solving optimization problems. Different possible configurations, or trajectories in this case, are termed solutions, and the candidate solutions symbolize a genome, usually represented by an array of bits. The candidate solutions form a population, and a portion of this population is stochastically selected depending on their fitness, which is determined by some objective function. The selected solutions breed by crossover to form new generations of offspring with a genome composed by inheritance, recombination and mutations. New generations are iteratively created a specified number of times, or until some satisfactory solution is generated. Usually, the initial population is generated randomly, giving rise to the potential of ending up with an optimal solution far from what would be imagined originally[12][16].

Other methods for programming the robots also exist, like the method of controlling the movements of a smaller copy of the robot, a teaching robot, making the real, larger scale robot repeat the movements[10].

Initially the lead through method was the most used, but currently over 90% of industrial robots are programmed using the teach method. The use of off-line programming is still limited, but increasing each year[3].

Especially for repetitive and high precision movements, the typical robot uses a teach pendant for programming. The robot is moved slightly, before saving the position with the teach pendant, and this step is repeated until the whole movement has been done. After the programming has been finished, the robot can repeat the whole movement at a significantly higher speed.

2.5 Simulation

In the process of design and production of mechanical parts, simulation software is often used in order to discover weaknesses in material or design before manufacturing solid parts. In this way, production cost can be reduced and design can be optimized.

A common simulation technique is the finite element method (FEM), a numerical technique for calculating stress and displacements of parts.

A finite element analysis simulation tool that utilizes FEM is offered by SolidWorks, the 3D CAD software tool used for designing KlimBot. SolidWorks is described in more detail in section 2.7.1 on page 12. Such simulation software has not been used for the project of this thesis, mainly because the strength of the layered material of 3D printed prototype parts with might be hard to simulate. Furthermore, the objective for this master's thesis has not been optimization, but rather determining the feasibility of the approach of a wall-climbing robot for bolted walls.

2.6 Master's thesis approach

The main idea by the approach of this master's thesis has been to keep things simple, as well as attempting to do as much practical research as possible, in order to get close to the stated main problem of the thesis, and earn sufficient experience for being able to determine whether or not the application of such a climbing robot is realistic and feasible.

The motto has been "learning by doing", gathering experience by experimenting, trying out and testing in a real-world environment, rather than by simulation, in order to be able to encounter realistic problems that might not be presented by a simulator. The focus has been on simplicity in order to make things work as early in the process as possible, and rather advance by reducing or removing certain assumptions and simplifications, such as the strict prerequisites regarding the climbing wall, along the way.

KlimBot has been designed with legs, because of the promising potential for a secure adhesion. Rotational actuators have been applied for enabling locomotion. KlimBot is designed to climb a predefined grid of bolts on a vertical and flat wall, where the bolts are separated by an equal distance both horizontally and vertically. This design approach is chosen in order for one of the most important challenges in the field of wall-climbing robots to be met, by attempting to make a reliable robot that can safely stick to the wall. By holding on to bolts that are already mounted to the wall, the risk of losing grip and falling down is highly reduced when compared to other adhesion methods.

KlimBot combines the benefits of a tracked robot, and those of a robot gripping to the surface. The requirement for bolts increases the potential for a highly reliable adhesion, when compared with the method of gripping on to the surface. It has some advantages over the tracked robot as it can move more freely over a surface, bolts are required to already be mounted onto the wall it's climbing, but the path it takes can be reprogrammed and adjusted as wanted. The possible abilities of climbing more complex structures and uneven walls as well as avoiding obstacles, could also make it superior to a tracked robot on certain occasions.

A major drawback with several of the adhesion methods mentioned is the weight restrictions they impose. The high potential of a secure adhesion by gripping onto bolts also give rise to the ability of carrying higher payloads.

If the most common programming method, the teach method, was the method applied for KlimBot, it would take ages for an operator to program the robots gait. The operator would have to move the robot to each single bolt. When imagining that the robot should be able to climb hundreds of bolts, it becomes quite clear that this method would not suffice. However, the dependency for a precise grid of bolts greatly reduces the required complexity of the control system. The climbing robot can be equipped with a preprogrammed climbing gait, which can be identical for each step between any of the bolts regardless of where on the wall the robot is situated, making it easier for an operator to give the robot instructions on where to go, as the desired move can easily be split up into a definite number of replayed predefined steps in either direction.

However, having to depend on a customized and designated path of bolts definitely reduces the area of use for the robot. Quite a few climbing robots have the ability of climbing walls and structures that were not necessarily designed for that purpose. A wall-climbing robot adhering to bolts might be less flexible in the sense of movement freedom compared to the wall-climbing robots that adhere directly to walls, but compared to a robot riding tracks, a comparison that is fear when considering the reliability of the adhesion, the freedom of movement is much greater.

The dependency for a specialized wall might be the main reason for the absence of previous research on wall-climbing robots for bolted walls. However, the potential for a highly reliable adhesion method combined with the potential of carrying heavy payloads sum up to a robot with a great potential for operation of a wide range of tasks by the addition of different tools, making the robot climbing bolted walls an interesting alternative and subject for research.

2.7 Tools and programs used

2.7.1 SolidWorks

The solid parts of KlimBot have been designed in the 3D CAD software tool SolidWorks. According to The Sheffield Telegraph, SolidWorks is the world's most popular CAD software[48]. Using SolidWorks, you usually start out drawing a two dimensional sketch as a base of the part you want to design, specifying the desired dimensions and relations between different components of your design. Then you transform your sketch into a three dimensional part by extruding your sketch from a selected plane. A rich variety of tools and features allows you to modify the extruded part to exactly match the design you have imagined. The ability to mate the parts you have designed together to form assemblies or larger parts, also in combination with imported parts from huge libraries or online communities, gives you the possibility to form complex models. The body part of KlimBot, seen in figure 2.1 on the next page, is created by combining a body base with four servo holders. As mentioned in section 2.5 on page 10, SolidWorks also provides simulation tools, allowing you to identify design constraints and errors in the design phase, saving you time and money by letting you create fewer prototypes. After creating all the parts for KlimBot, and combining them to a complete assembly, the suitable dimensions for the parts in my design were determined, as well as the distances between the bolts on the wall for KlimBot to climb. SolidWorks also makes it possible to create images, drawings and detailed documentation of your designs. After finishing the design, SolidWorks was used for creating STL files of KlimBot's parts. STL is a format originally used to hold the information needed for Stereolitography machines[37], that now is the standard file type used by most additive manufacturing systems. An STL file holds a triangulated approximation of a 3D model´s surface - a complete listing of the vertices and normals for all the triangles describing the 3D object, represented as coordinates in a three dimensional Cartesian coordinate system. In figure 2.2 on the next page you see an STL representation of KlimBot's body part.

2.7.2 Insight

The Insight software[44] prepares the STL output from SolidWorks for 3D printing by optimizing build orientation, slicing, creating support structures and generating material extrusion paths. Slicing is the process of cutting the 3D model into thin horizontal layers and planning the toolpath for each layer. Generally, the thinner the layer, the higher resolution and the smoother the surface on the printed part gets. Each layer is divided into two parts - shell and infill (represented by red and green lines in figure 2.3 on page 14). The shell makes a solid and strong outer perimeter, the infill is the amount of filling within these perimeters. Lower infill percentage gives a lighter object, higher infill percentage makes a stronger object. Support structures are needed for dealing with gravity, to prevent material falling

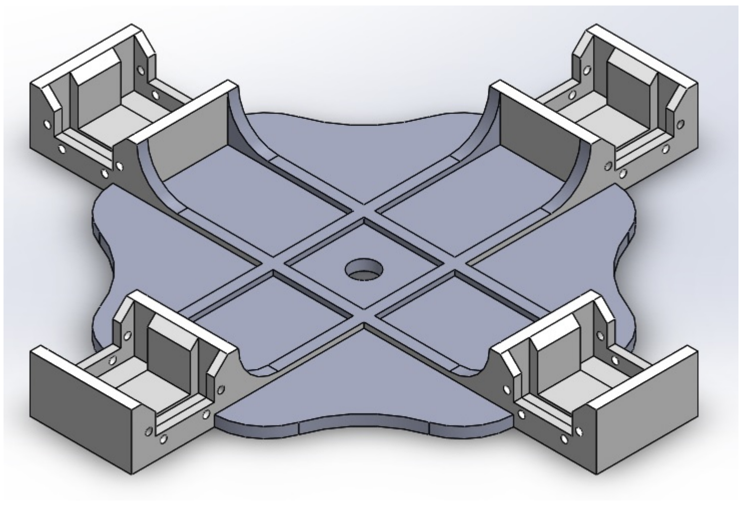


Figure 2.1: KlimBot body designed in SolidWorks

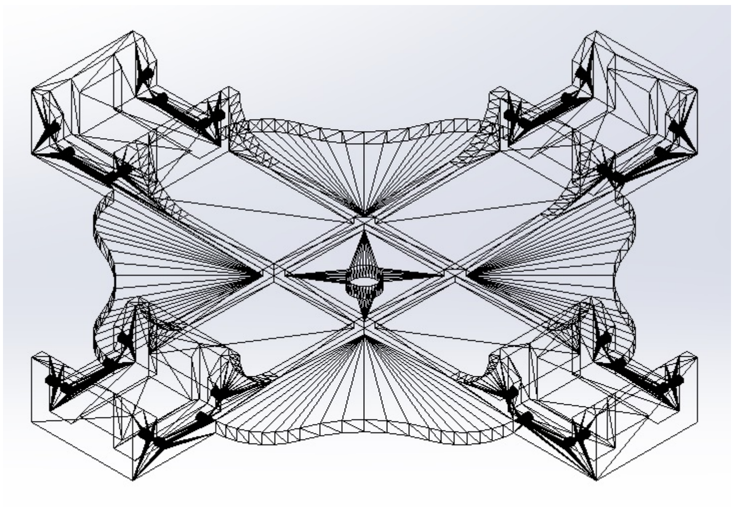


Figure 2.2: KlimBot body as represented in an STL file

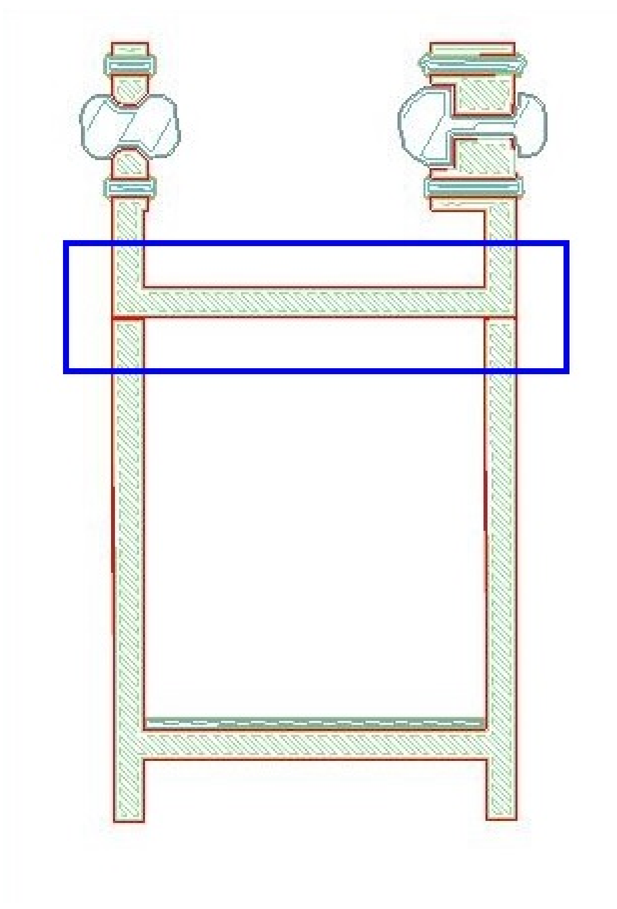


Figure 2.3: Insight capture showing a layer of a KlimBot leg with delamination between to parts, highlighted by a blue square.

down or through hollow features. Insight lets you customize support structures for best possible material use and easy removal. Small horizontal holes with diameter less than 3-4mm can be printed without support, as removing support from these holes would be very difficult. Build speed is influenced by several variables like the amount of support material, layer height and infill percentage. As seen in figure 2.3, Insight is also helpful for finding design flaws, in this case delamination between two parts caused by incomplete mating.

2.7.3 Fortus 250mc 3D Printer

Manufactured by the American company Stratasys[47], the Fortus 250mc 3D printer utilizes the Stratasys patented Fused Deposition Modelling (FDM) technology. FDM is an additive manufacturing process, in which three dimensional objects are created by successively adding layers of material under computer control, building a part from nothing[14]. In contrast, subtractive manufacturing processes like turning and milling uses a block of material and removes excess material until the desired object is created. Additive manufacturing processes are generally more

Figure 2.4: Fortus 250mc 3D Printer[47]

time and cost effective, at least when it comes to single prototypes and small batches, and they provide the ability to create more complex geometries[17]. Although some new additive manufacturing processes can create parts in materials such as metal, almost all additive technologies use different sorts of plastic for creating parts. For metals, wood, foam and other materials, subtractive manufacturing processes are the most likely technology used[21]. FDM produces parts by depositing thermoplastic material through a heated nozzle. A thermoplastic filament is fed to the nozzle from a coil, and the heat makes the thermoplastic soft and moldable, before it returns to its solid form when cooled. The nozzle is moved both horizontally and vertically by a numerically controlled mechanism to follow the toolpath provided by the Insight software. The size of the parts created by any FDM manufacturing machine is limited by its build envelope. Fortus 250mc has a build envelope of 254mm * 254mm (base) * 305mm (height). The most common plastics used for FDM are different variants of Acrylonitrile Butadiene Styrene (ABS) and Polylactic Acid (PLA). ABS has lower density (is lighter) than PLA. PLA is harder and more rigid than ABS, but also more brittle[9]. Fortus 250mc uses ABSplus-P430[46], a material with greater tensile, impact and layer bonding strength than standard ABS. ABS is more heat resistant than PLA, but also tends to warp more easily during cooling. Warping can be reduced by using a heated printer bed, or even better by heating the entire chamber, creating a thermally isotropic build envelope, the latter being the case for Fortus 250mc. Thermoplastics allow a certain degree of overhang, but support structures are usually required during the printing process. Post processing of the printed parts involve removing these structures, usually by breaking away the support material. There are also soluble support material available, that dissolve when placed in a chemical bath with high pH value (basic). This usually takes quite a long time, but the process is hands-free and also removes hard-to-reach support material.

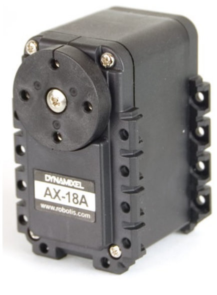


Figure 2.5: Robotis Dynamixel AX-18A Robot Actuator[35]

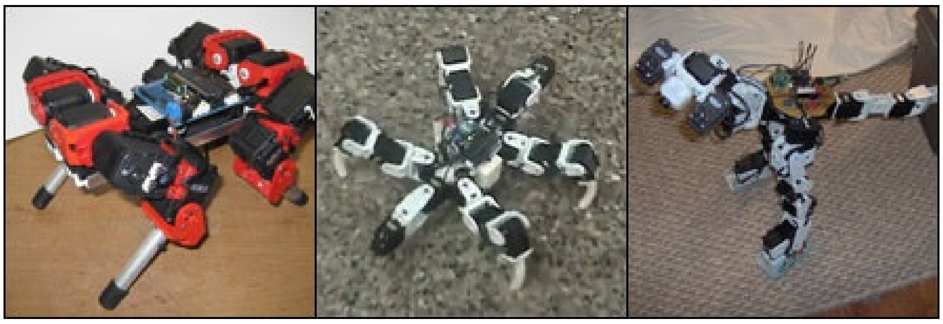


Figure 2.6: Projects using Dynamixel servos[49]

2.7.4 Robotis Dynamixel AX-18A Robot Actuators

Dynamixel AX-18A[35] is a high performance rotational actuator, with the ability to track its own speed, temperature, voltage and load. Rotational actuators with position feedback mechanisms are also called servomotors or servos. Dynamixel servos allows relatively high speed, up to 97 rounds per minute without load, and has a stall torque of 1.8 Nm. Both speed and torque can be set in 1024 increments. The servos can be set to operate in a free running wheel mode or joint mode with a running degree of 0 - 300 in 1024 increments, giving a positioning precision of approximately 0.29 degrees. KlimBot uses eight Dynamixel servos, two for each limb. These Dynamixel servos have been chosen for their relatively high torque, good precision and feedback functionality at a fairly reasonable price. Robotis offer a wide range of Dynamixel servos at a wide price range, with different speed and torque properties, as shown in figure 2.7 on the facing page. These are also some of the main reasons for why different Dynamixel servos are popular robot actuators. Some projects can be viewed in figure vrefdynaProj. All of the servos can be assigned an unique ID and are linked together in a daisy chain. Serial communication with the servos, with a baud rate of up to 1MB, is established with the USB2Dynamixel 3-wire bus communication dongle seen in figure 2.8 on the next page, a device used to operate Dynamixel servos directly from a computer. It is also possible to carry out communication without being wired to a computer, by wireless communication through a Zigbee module, or by connecting a microcontroller locally on the robot. The main controller

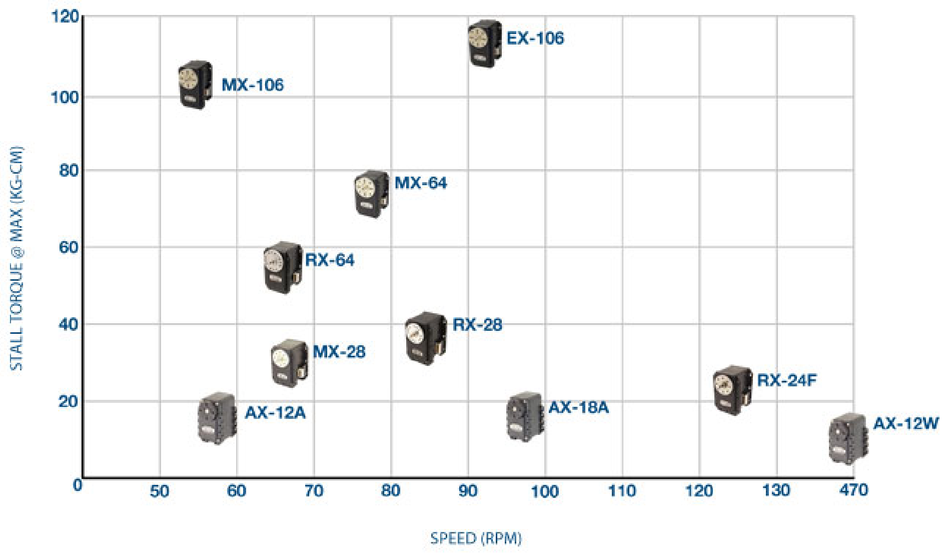


Figure 2.7: Strength & Speed Chart of the Dynamixel Family[50]

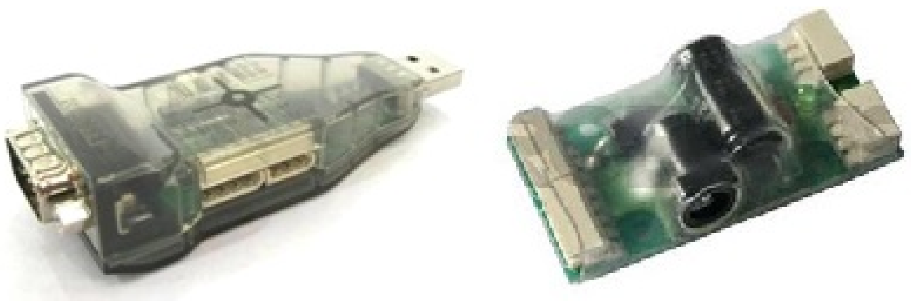


Figure 2.8: USB2Dynamixel dongle and SMPS2Dynamixel adapter[36]

communicates with the servos by sending data packets called instruction packets, addressed to a specified servo by ID or by broadcasting to all the servos in the daisy chain. The servos respond by returning status packets. Power is supplied by connecting a 12V power cable to a SMPS2Dynamixel adapter (figure 2.8), which is also linked to the servos in a daisy chain. Alternatively, power can be supplied by attaching a battery pack locally on the robot. Behaviour is controlled by directly writing values to the internal registers of the servo. The servos send some feedback information automatically as a response to the behaviour alterations, but information can also be retrieved by reading directly from the servos registers. Some data, like speed, goal position and torque limit, is stored in RAM and is reset to its initial value whenever power is turned on, whilst other data, like ID and angle limits, is stored in EEPROM and is kept even if power gets turned off.

2.7.5 Processing

Processing is a development environment and also a Java based programming language, originally created for teaching computer programming fundamentals through visual feedback. Processing projects are called sketches, and graphical output of these sketches is drawn in a display window. Processing has been used because of its serial library which makes it capable of communicating with the serial/COM ports of the computer running it. All instructions and communication with KlimBot has been written in Processing.

2.7.6 NetBeans

NetBeans is another development environment, primarily for programming in Java, but it also supports other languages. NetBeans has been used for programming the KlimBot Control System and to create the graphical user interface. The Processing project has been included in the NetBeans Java application, meaning communication with KlimBot can be carried out simply through interaction with the KlimBot Control System application.

Part II

The project

Chapter 3

KlimBot design

3.1 Initial design ideas

Initially, two different design ideas unfolded, inspired by previous approaches described in literature, as well as the Walloid project at the University of Oslo[23]. Both of them were thoroughly designed. Evaluating two different designs provides the opportunity of using comparison as a tool for creating the most applicable design. Both designs describe legged robots with a base part connected to four extremities consisting of an inner and an outer part. The individual design components have been named as follows: the base part was named *body*, the inner extremity part closest to the body *thigh* and the outer extremity part *leg*. The extremities will be referred to as *limbs*. All the parts are connected by Dynamixel servos acting as revolute joints. In the following text, the two designs are presented, along with arguments explaining the choice of design that was improved and realised.

3.1.1 Design A

Design A can be viewed in figure 3.1. The design is biologically inspired, and its appearance bears resemblance to a spider, although it only has four legs. The objective has been to make a functional robot, as simple

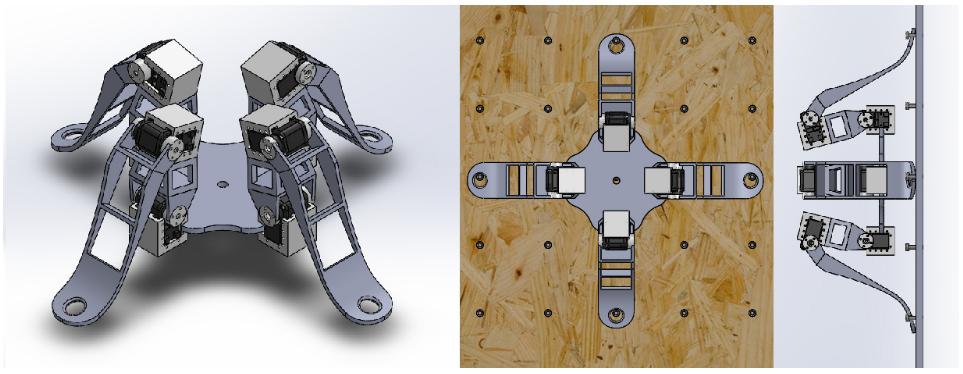


Figure 3.1: Design A

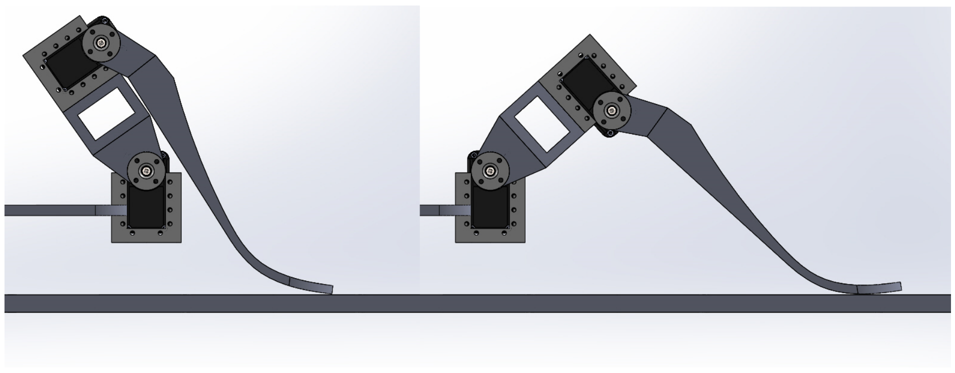


Figure 3.2: Limb in retracted and outstrecthed position

as possible. Even though eight legs provide stability and robustness, the complexity of a control system for an eight legged robot is drastically increased compared to that of a four legged robot. Four legs also suffice for traversing a grid of bolts, by stretching out its limbs the robot can reach bolts upwards, downwards, left and right.

The design has quite a compact body, the body base has a diameter of 150mm. This size was chosen to allow a light body as well as providing the required space for when the thighs are in the retracted position. The body base also serves as a platform where additional features may be placed if desired, such as a microcontroller, power supply or small tools. The hole in the center of the body was intended for giving the ability to feed a wire from a battery pack or something else from the underside of the body.

The outstretching movement is largely carried out by the thigh, as seen in figure 3.2. The length of the thigh has been chosen with the aim of making the step length sufficient without compromising body stability and adhesion; increased thigh length results in increased range, but also increases the diameter of the rotational trajectory elevating the leg from the wall, thus increasing the angle in which the leg connects to the wall.

The leg has been designed with a curved end to give a smooth, rolling movement when the angle between the leg and the wall is altered during locomotion. The curve is also meant to give a secure grip on the bolt by minimizing the angle difference in which the leg connects to the wall when being retracted and outstretched(figure 3.2). Furthermore, the curved shape helps distributing the strain applied to the leg when pushing against the wall. Where curved, the leg is solid for extra strength. As step length primarily is determined by thigh length, the legs are designed long enough for benefiting from the curved shape, but as short as possible for keeping the legs close to the body, thus reducing the torque needed for locomotion.

The gripper is simply a hole at the curved part of the leg. Adhesion is obtained by placing the hole over the bolts in the wall. This simple gripper allows for locomotion by both pushing (lower leg) and pulling (upper leg). Gripping the bolts in this fashion provides some necessary flexibility; adhesion is maintained when the orientation of the leg is slightly altered during locomotion, as described above. Without this flexibility, complex

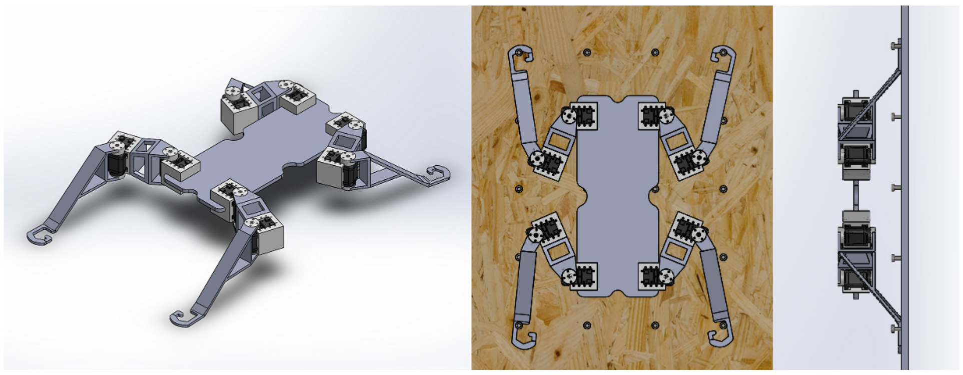


Figure 3.3: Design B

calculations would be required for keeping the leg motionless, and the body part would no longer have the ability to keep a constant distance to the wall during locomotion.

3.1.2 Design B

The alternative design can be seen in figure 3.3. Its appearance resembles a human climber more than a spider. The limbs are not centred as in design A, but rather placed on the corners of the body base, similar to the human shoulder and hip joints. In this way, the limbs are able to stretch out to facilitate locomotion in two different directions. Two limbs are stretched out in the desired direction of movement.

The body base has a 120mm * 300mm rectangular shape. It is designed in this fashion in order to provide enough space to prevent the thigh parts from colliding with each other when in the retracted position.

The thigh part for design B is equal to the thigh part of design A, although a mirrored version had to be designed for two of the limbs, for left and right orientation. As for design A, the length design B is able to stretch out is largely decided by the length of the thigh.

The legs have been designed with sharp edges, to maximize the contact area between leg and wall. This could be done since the leg always has the same angle as to the wall. As with the thighs, mirrored versions had to be designed.

The gripper is designed as a hook. A simple hole wouldn't suffice, as the gripper has to slide on to the bolts for adhesion. The opening of the hook has to be large enough to enable sliding over the bolts, while at the same time being small enough to secure a safe grip. The gripper also has an indentation which is used for adhering the lower legs to the bolts.

3.1.3 Design choice

Both designs are vertically and horizontally symmetric. For both the robots, all their limb parts are principally equal, mostly for aesthetic reasons. As the upper and lower legs serve slightly different purposes (pulling and

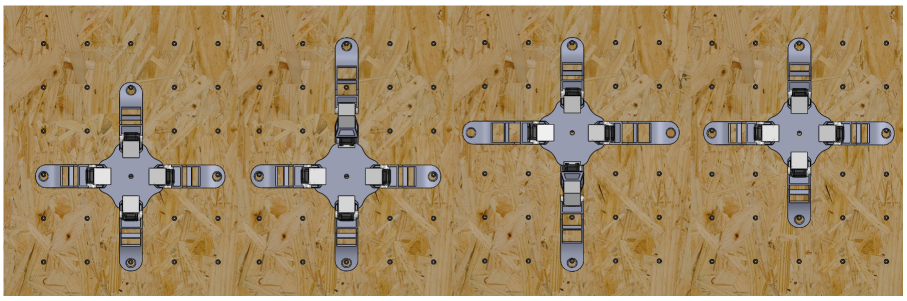


Figure 3.4: Climbing gait for design A

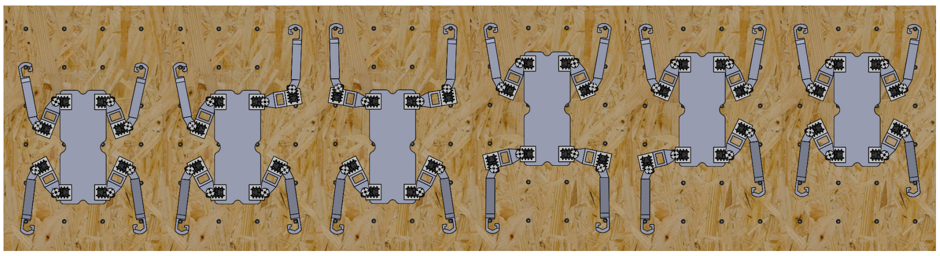


Figure 3.5: Climbing gait for design B

pushing), they might have benefited from specialized designs, especially in the case of design B, where the legs might lose grip when pushing off the bolts during locomotion.

The main difference between the two designs is the orientation of the servos. For design A, the servos are standing up, making their rotational axes parallel to the wall, whilst design B has its servos lying down, making the rotational axes perpendicular to the wall. This makes the design A able to lift its limbs off of the wall, whilst design B provides the ability to move its limbs more freely over the surface of the wall.

It would have been interesting to follow both designs further down the line before making a choice between them, but at this point it was time for deciding which design to improve and realise. The following arguments made design A the preferred option. However, after having acquired more experience through a lot of work and research on KlimBot during this thesis, the choice of design might have been different. This is further discussed in section 6.1.1 on page 53.

As design A makes it possible to raise and lower its limbs with respect to the wall, it could make the robot able to climb slightly bent walls, which is required when climbing surfaces such as nuclear storage tanks, as mentioned in section 2.2 on page 5. Design B drags its legs upon the surface, making it vulnerable even for small cracks or bumps. The friction between the wall and the legs would also lead to wear on the robot parts and possibly also the wall.

For design A, its lower limb serves as a passive tail, which is beneficial for balance and stability. For climbing robots, the center of gravity creates a destabilizing moment that pulls the upper part of the robot away from

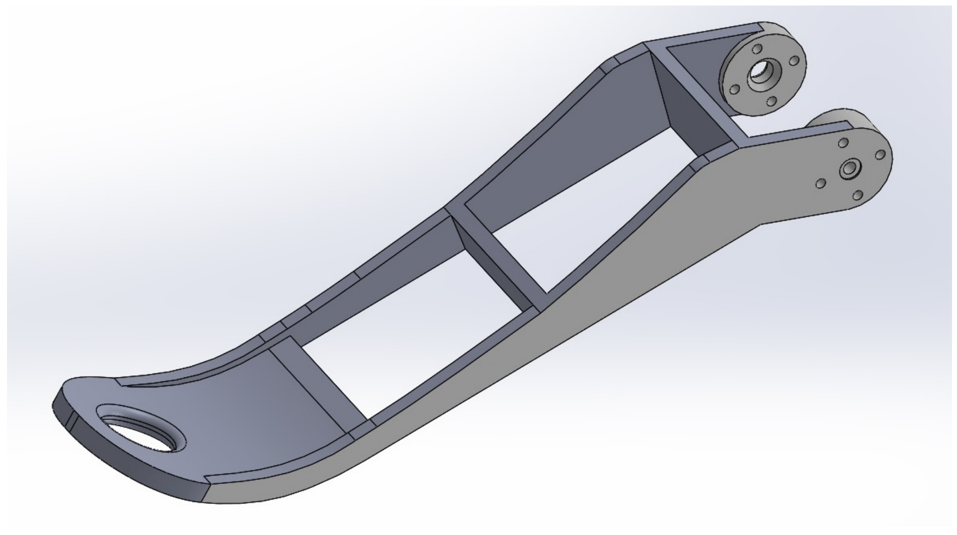


Figure 3.6: Modified KlimBot leg part with strengthening structures and reduced thickness

the wall. A tail laying against the wall could be used to minimize this destabilizing moment[51]. For design B to have a tail, it would have to be designed exclusively for this purpose.

Design A seemingly had a potential of climbing at a higher velocity than design B, since its climbing gait only used two limbs for locomotion, hence requiring few operations for each step, as illustrated in figure 3.4 on the preceding page. Design A simply stretches out, releases two grippers, moves, fastens the two grippers, and retracts the lower leg. A climbing gait for design B is illustrated in figure 3.5 on the facing page, and shows an apparent increase in the operations required for each step, resulting in a slower design. However, an optimized climbing gait for design B might contravene this argument, and is amongst the talking points for the mentioned discussion in section 6.1.1 on page 53.

Additionally, the symmetry of design A means it could be placed on the wall in any orientation and it would still look the same. If equipped with a gyroscope, it could automatically determine which way was up and which way was down. Imagining the robot could be able to move from one wall and onto another, a completely symmetric robot would not have to turn 180 degrees before mounting the second wall, it could simply redefine its lower limb to act as its upper limb, and vice versa.

3.2 First assembly

Having decided the main design idea, some modifications on the design were done before printing out the parts and assembling the first version of KlimBot.

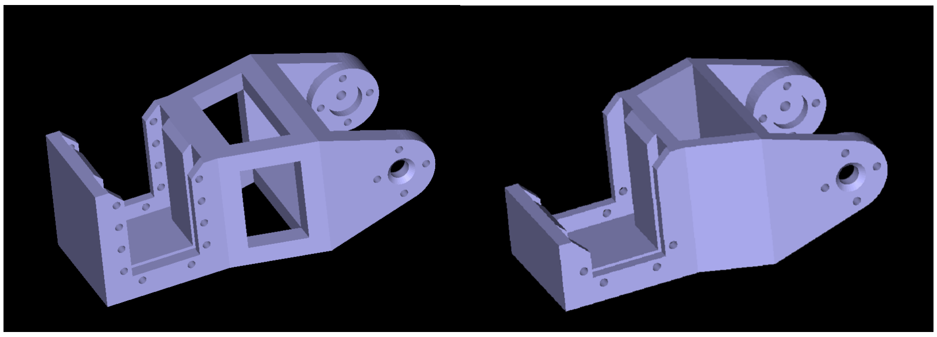


Figure 3.7: KlimBot thigh part before and after design modification

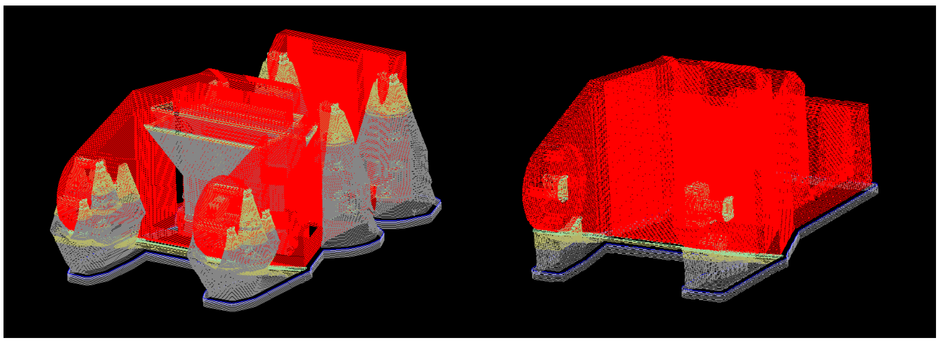


Figure 3.8: Support structure (yellow and gray) required before and after modification

3.2.1 Modifications

After presenting the design to a senior engineer a few problems concerning practical design were discovered. These practical problems were dealt with by returning to the drawing board to improve the design.

Some of the modifications were made for reducing weight, and some for strengthening the parts. Thickness was reduced from 6mm to 4mm, and the height of the servo holders was reduced. Strengthening structures were designed on the body base and on the curved part of the legs, as seen in figure 3.6 on the preceding page. Furthermore, some of the holes in the servo holder was removed, as the screws through the lower holes would be sufficient for securely fastening the servos to the holders. The thigh parts were designed slightly shorter, in order to reduce the rotational arch and consequently the angle difference between leg and wall when in retracted and outstretched position. All these modifications were made to make the design as light and sturdy as possible.

Other modifications were made for reducing the amount of support structures required, thus reducing production cost and the post processing required. The body base was lowered to level with the servo holders, as vast amounts of support material would be required to raise the body base from the 3D printers bed. This also slightly lowered KlimBot's center off mass, bringing the center of mass closer to the wall when KlimBot is attached. 3D printing by FDM technology involves successively adding

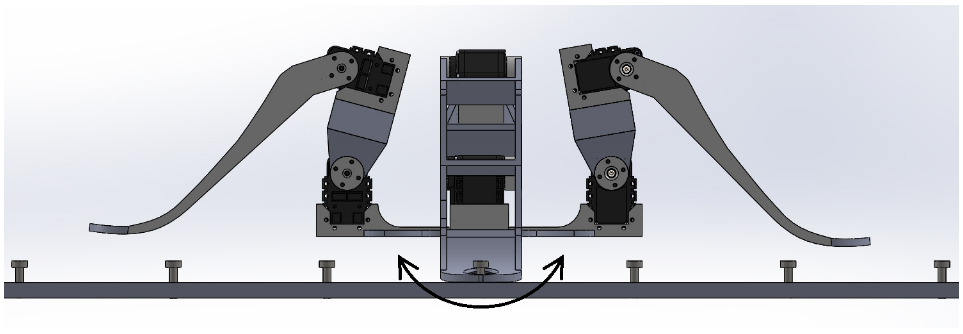


Figure 3.9: Unsteadiness caused by rounded tip of leg

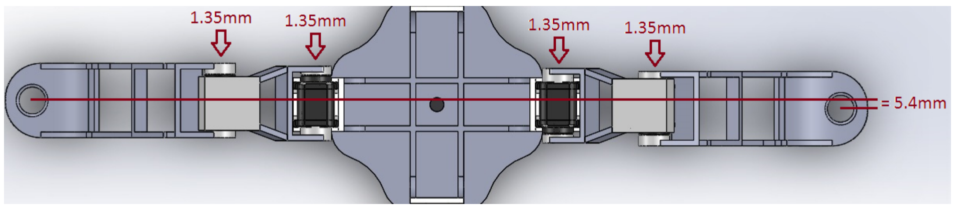


Figure 3.10: Skewness propagating through each joint

thermoplastics in layer after layer. The first design of the thigh parts had overhanging structures, meaning a considerable amount of support structure was needed. The outer surface of the thigh part was made solid to deal with this. A comparison of the thigh designs can be viewed in figure 3.7 on the preceding page. Further support structure reduction was done with the Insight software, and in combination with the physical design modifications, support structure requirements was drastically reduced, as seen in figure 3.8 on the facing page.

3.2.2 Challenges

During the test process, which is described in chapter 4 on page 33, some design challenges became apparent regarding this first assembly.

The rounded tip of the leg part made KlimBot wobbly and unsteady, most noticeably during locomotion, when two of KlimBot's legs released their grip to the wall, and only two legs had contact with the surface of the wall. As illustrated in figure 3.9, the rounded tip made the contact area between KlimBot and the wall insufficient for stability.

There was also some trouble when wanting to retract the limbs further than the current design allowed, as the leg part would collide with the thigh part.

Another design problem originated from erroneous dimensions in the SolidWorks model of the Dynamixel servos, leading to skewness. The components for connecting servos to the solid parts were in consequence designed with incorrect dimensions, meaning the solid parts got an offset off 1.35mm with respect to the base of the servos. As there are four of these connections (two limbs with two servos each) along both axes of

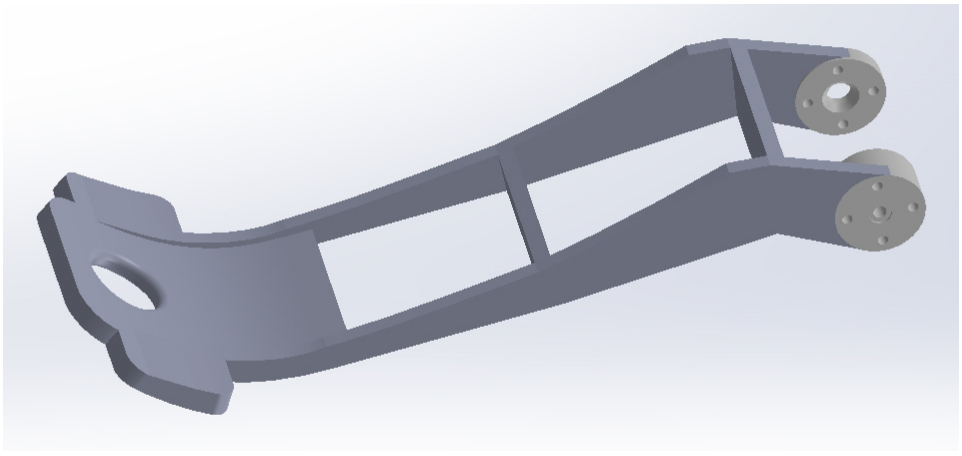


Figure 3.11: Leg design for improved stability

movement, the total offset became 4 * 1.35 mm = 5.4mm, as illustrated in figure 3.10 on the previous page. This results in quite a substantial skew when considering that the bolt heads on the wall have a diameter of 10.32mm, and the hole of the gripper has a diameter of 19mm.

3.3 Improved assembly

For dealing with the design problems presented in section 3.2.2 on the preceding page, further design modifications were made before printing the parts and reassembling KlimBot.

3.3.1 Modifications

In order to increase the stability of KlimBot, two modifications were done to the design of the leg part. The previously rounded tip was straightened, and the end of the leg part was widened by extrusion on both sides, effectively increasing the area of contact between the leg part and the wall. Both modifications can be viewed in figure 3.11.

For being able to decrease the angle between leg part and thigh part further, the leg was designed thinner close to the servo connectors, as seen in figure 3.12 on the facing page, where the new design has been paired with the older one for comparison. Bringing the leg closer to the thigh meant the distance between the grippers of the two legs of the same axis of movement could be reduced, resulting in a tighter grip on the bolts. Furthermore, when designed this way, the thigh can be retracted more than previously, giving a better angle between leg and wall compared to what was achieved by the previous design, for corresponding distances between the grippers.

To avoid skewness, the dimensions of the connection components of the limb parts were modified accordingly, as seen in figure 3.13 on the next page.

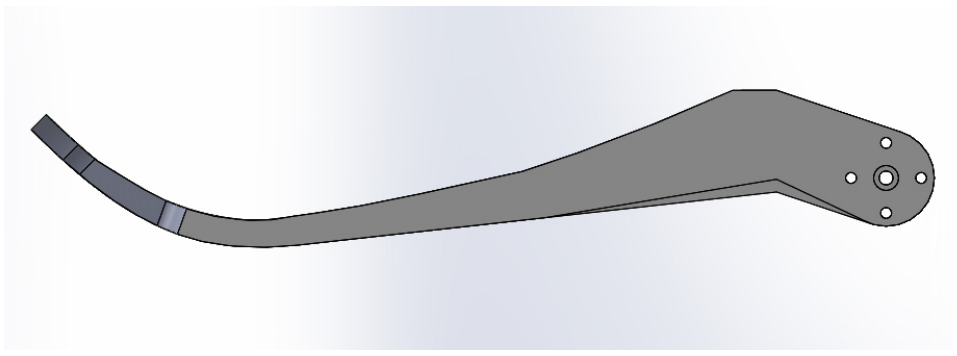


Figure 3.12: Thinner design of leg

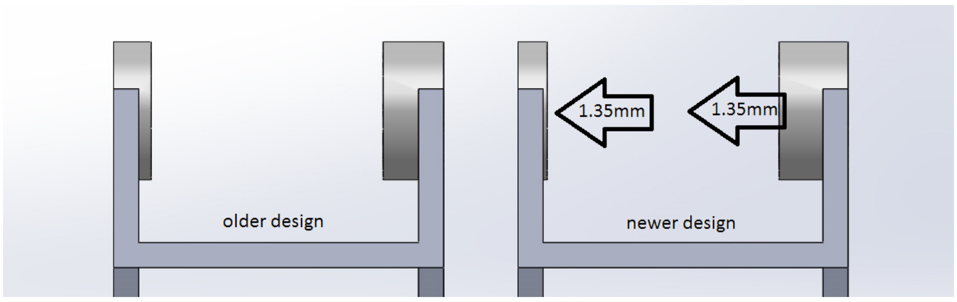


Figure 3.13: Reducing skew by modifying servo connection components

3.3.2 Challenges

One of the major challenges for a wall-climbing robot is reliability; acquiring a secure adhesion to the wall. The simple adhesion method of KlimBot, simply placing a hole over the bolts, might not necessarily satisfy this criterion. On some occasions during testing, the upper leg lost its grip as the gripper slipped off of the bolt it was holding on to.

3.4 Final assembly

3.4.1 Modifications

To improve the reliability of KlimBot's adhesion, a rather small, but efficient, adjustment was made to the gripper hole of the upper leg. An additional, smaller hole was introduced, dimensioned in such a way that it has a larger diameter than the bolt, but at the same time smaller than the bolt head, enabling the leg to slide over and lock onto the bolt more securely, as illustrated in figure 3.14 on the following page.

The final assembly of KlimBot can be viewed in figure 3.15 on page 31.

3.5 Further possible improvements

Additional improvements may still be made by modifying the current design, and a couple of these possible modifications are suggested in the

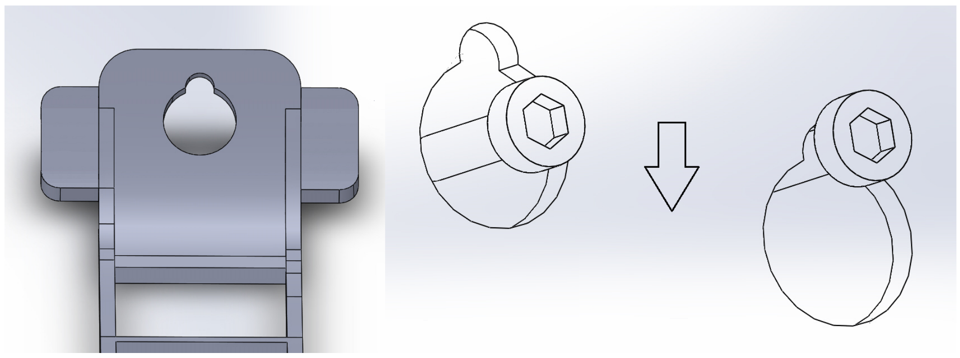


Figure 3.14: New design for more secure adhesion

following text.

Initially, the leg parts were all identical, although they have slightly different tasks, and performance could be improved by specializing the design for each leg, accordingly. The modification of the gripper hole of the upper leg considerably increased the reliability of KlimBot's adhesion, and similar alterations to the remaining gripper holes could also prove to be beneficial.

For reducing the risk of a leg getting stuck to the bolt head when releasing its grip, KlimBot should be able to lift itself slightly higher than what it is capable to with the current design. Further elevation is limited by the upper limbs capability of retraction. To accomplish further retraction, the leg part could be designed even thinner by the servo connections. Alternatively, a steeper curve could be applied to the upper part of the leg.

Slightly bigger gripper holes could also be a good idea, as this would reduce the precision requirements for the climbing gait by allowing a larger margin of error.

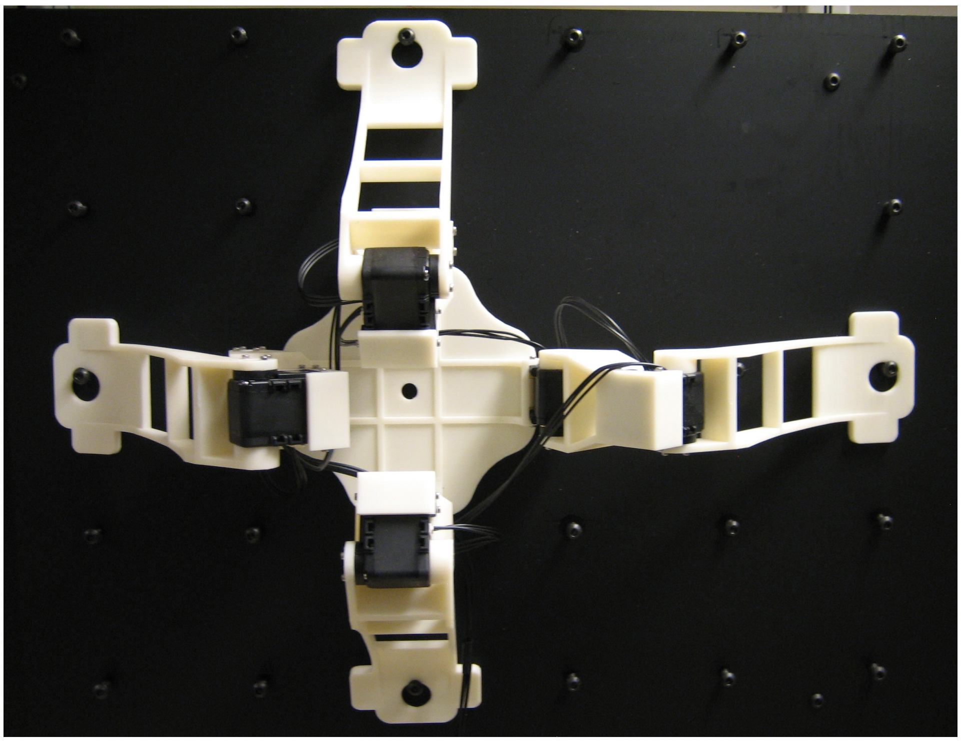


Figure 3.15: The final version of Klimbot

Chapter 4

Testing

4.1 Wall

In order to test the functionality of KlimBot, a bolted wall for climbing had to be designed and built.

4.1.1 Test wall

The first wall created for KlimBot was made out of a plywood board attached to a wheeled and rotatable metal base, providing the ability to alter the steepness of the wall, as seen in figure 4.1. The dimensions of the wall are 80cm * 110cm. 54 bolts are screwed into the wall giving a square grid of 6 * 3 *reachable bolts*. Reachable bolts are defined as the bolts that KlimBot can securely attach its upper leg to, whilst also being able to grip on to supporting bolts with the remaining legs. Furthermore, KlimBot has to be able to travel to the reachable bolt from its current position.

All bolts have an equal distance of 100mm between each other in both horizontal and vertical direction. This distance was decided when dimensioning the solid parts in SolidWorks. The bolts used have a diameter of 6mm, whilst the bolt heads have a diameter of 10.32mm. The bolts are screwed into the wall so the distance between the bolt head and the wall is

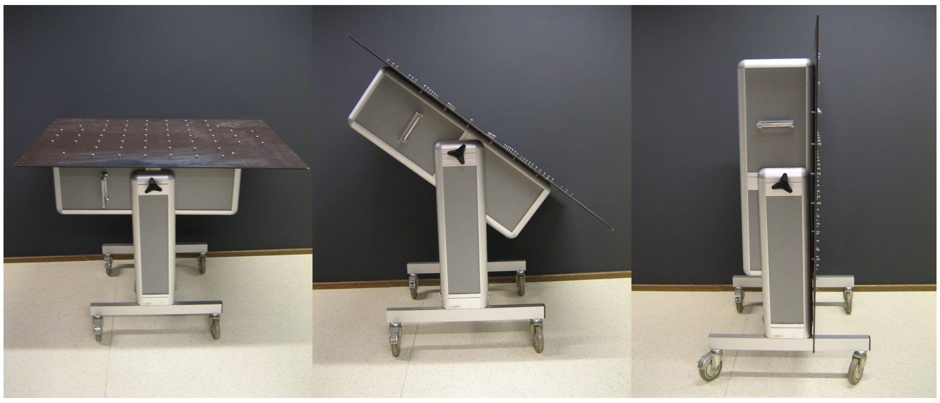


Figure 4.1: Different configurations of the test wall

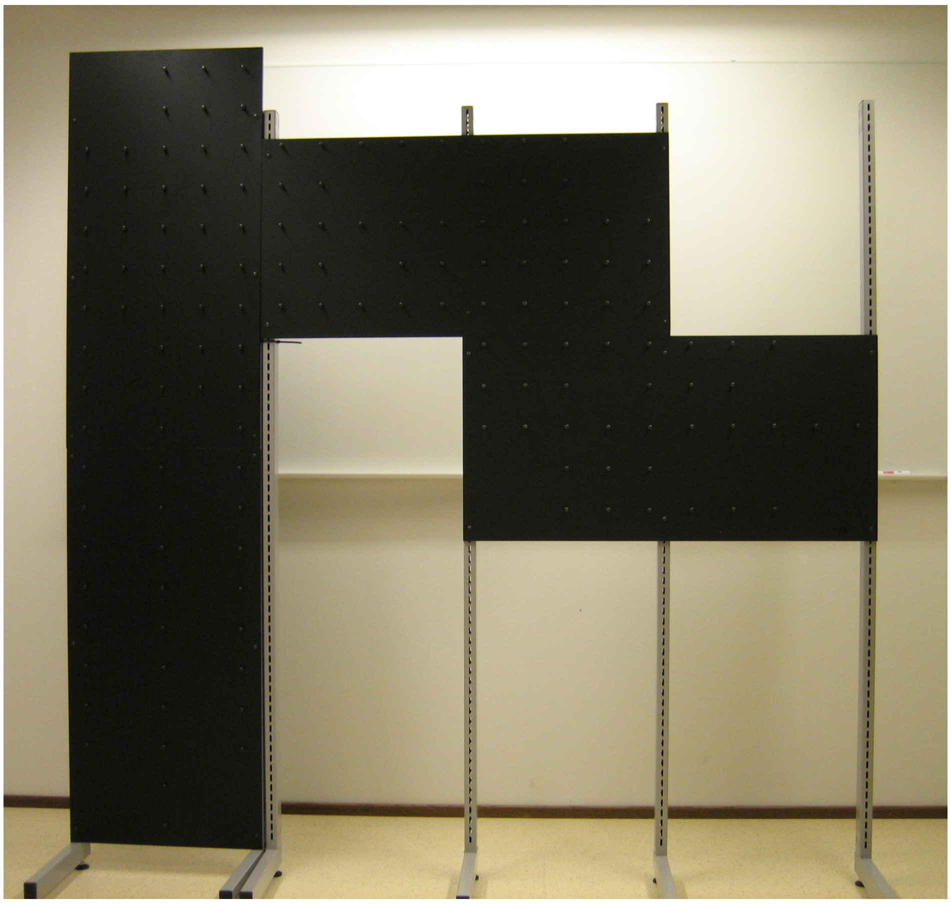


Figure 4.2: The demonstration wall

10mm.

As mentioned earlier, one of the main challenges for a climbing robot is reliability; securely adhering to the wall. Being able to rotate the wall to alter the steepness of the wall was of great importance, especially in the early stages of testing. KlimBot's climbing gaits were initially tested with the wall in a horizontal position. This way, the risk of loosing grip and falling off the wall was drastically reduced. Although both the Dynamixel servos and the solid parts of KlimBot are fairly impact resistant, falling from the wall would most likely result in damage. Replacement of damaged parts could prove to be expensive both in cost and time. As work with KlimBot progressed, and reliability was improved, the steepness of the wall could be increased gradually, until finally KlimBot was climbing a vertical wall.

4.1.2 Demonstration wall

Later in the process, a more challenging wall was built in order to get closer to answer the main problem of the thesis. In addition, the materials used provides a more representable wall for demonstration. The wall was constructed by connecting four black plastic boards measuring 100cm *

50cm to six legged girders. The total dimension of the wall is 200cm * 200cm. 143 black bolts are screwed into to the wall, giving a total of 52 reachable bolts, almost tripling the count of the test wall. However, the main thought behind the design of the demonstration wall was to create a longer optimal path between the two farthermost bolts. The longest optimal path on the test wall requires seven steps, whilst the longest optimal path for the demonstration wall requires thirty-three steps, almost five times as many as for the test wall. The demonstration wall can be viewed in figure 4.2 on the preceding page.

4.2 Processing sketch

In order to communicate with the servos, a Processing sketch was created. The USB2Dynamixel dongle contains drivers that create a virtual COM port, enabling serial communication through a computers USB port. Processings Serial library was imported to the sketch, and communication was initialized by specifying the virtual COM port number and the desired baud rate. Manipulation of the servos behaviour is achieved by directly modifying the values stored in the internal registers of the servos. However, a simplified abstraction was obtained by importing the SimpleDynamixel library [24], which provides high-level communication functions.

With the processing sketch, all servos were initiated with starting positions and moving speed. Different functions for carrying out the desired communication and test functionalities were written, most of which were called by entering keyboard input. In this way, instructions concerning speed and goal positions as well as enabling and disabling of the servos torque could be given to KlimBot during runtime. Additional functions for obtaining information such as the servos current positions and printing them on screen were also written.

During the test phase, the source code of the Processing sketch was frequently modified, and short command sequences were gradually created and combined until the sketch finally included complete sequences of commands to all the servos for moving KlimBot in all four directions.

4.3 Programming approaches

To test KlimBot's climbing gait, the Processing sketch had to be programmed with the positional instructions to send to the servos in order for the actual movements to be performed. Thus, the objective was to determine goal positions for the parts of KlimBot, and sequentially determine the position values of the servos that lead to the specified goal position. Two different approaches for accomplishing this were considered.

- *Calculation - inverse kinematics.* Kinematics is the use of geometric calculations for describing and determining the motion of points, objects or groups of objects. An assembly of solid parts connected by joints is often referred to as a kinematic chain. Forward kinematics is the technique of, when given the joint angles, calculating the position of a kinematic chain's end effector (in this context, the gripper hole on KlimBot's leg). In contrast, inverse kinematics is the technique of calculating the joint angles that lead to a given position and orientation of the end effector. These calculations are the basis for enabling programming with a teach pendant, that allow movement in a number of different co-ordinate systems. After acquiring the desired joint angles, these could by translated to servo position values.

- *Reading - lead through method.* As described in section 2.4.1 on page 8, when using the lead through method, the robot parts are physically moved whilst saving the positions of the joints along the way. By disabling torque for both servos on one limb at a time, that limb could be physically moved by hand in small steps, before the current position of the servos were read from their registers and saved. For each position saved, the relative difference between the two servos change in position was calculated, and the speed for each servo was set accordingly, in order to create a fairly smooth movement.

The latter approach was preferred, as it was reasonably straightforward, and gave the possibility of quickly starting the actual testing. Simply moving KlimBot's parts before reading and saving the servos positions provided an efficient way to get started. Determining the servo settings for acquiring a desired end effector position with inverse kinematics would require relatively complex calculations. Using inverse kinematics, the desired position of the end effector has to be represented by coordinates. When testing to find a satisfactory gait, the goal positions are not necessarily known in advance. Furthermore, the objective was not to create a teach pendant for an operator to be able to dictate small movements, but rather to define a functional climbing gait for KlimBot, so the operator could easily make KlimBot cover a considerable distance. Additionally, the lead by method corresponds to the method practice applied throughout this thesis: learning by doing - trying it out.

However, programming the gait by lead through method proved to be a very time-consuming task, and the choice of approach might have been different if the experience acquired during testing and programming was present at the time. The choice of programming approach is further discussed in section 6.1.2 on page 55.

4.4 Test process

4.4.1 Servo testing

Initially, some testing was done on the Dynamixel servos, without them being connected to the solid parts of KlimBot. All eight servos were first tested individually, and given an unique ID, before they were connected together in a daisy chain. Miscellaneous instructions for manipulating servo behaviour were tested, both by addressing the servos independently by ID, and by broadcasting instructions to all servos. Instructions for reading the feedback functions were also tested. The angle limits for all servos was determined and saved to ensure they would not be exceeded when mounted to KlimBot, since forcing two parts against each other could lead to self-destruction. The servo speed was also set very low in order to have reasonable overview of all the servos during testing. In the Processing sketch, servo positions have been set directly, but speed has been set as relative speed, so that a change in KlimBot's normal speed only has to be modified at one place in the code. Finally, all eight servos were assembled with the solid parts of KlimBot, and testing on KlimBot as a whole could commence.

4.4.2 Stretching out

As mentioned, the early stages of testing were done with the wall at a low angle, to prevent KlimBot from falling off the wall during testing. For the gripper holes to actually act as grippers as a result of gravity, the angle of the wall was quite early set to approximately 45 degrees. The first goal set was to make the upper leg release its grip of the bolt, stretch out towards the next bolt, and hook onto that bolt. Torque was disabled for the two upper servos, and the upper limb was physically moved in small steps before reading the position values of both servos for each step. Then each single step was tested separately by writing the goal positions to the Processing sketch and running it. Unfortunately, this process was not as precise as expected. As torque for the servos was entirely disabled, the limb had to be held up by as well as being moved to the desired position, so the lack of precision with this method was most probably caused by a less steady hand than required, in combination with the fact that gravity did not act upon the servos in the same way it would have done without human interaction. Considering that each position value increment translates to approximately 0.29 degrees, it proved difficult to achieve the exact position desired with this approach. However, it gave approximations for the desired positions, which were appropriate as references for further improvement. Functions for incrementing and decrementing the position values of the servos were written to the Processing sketch, in order to improve precision by performing small adjustments during runtime, before reading out the new positions. In this fashion, the first goal was reached, as the upper leg was successfully programmed to release its grip, stretch out and place its gripper hole over the next bolt.

4.4.3 Releasing side grippers

The next step was to release the side grippers in order for KlimBot to be able to drag itself up to the next row of bolts. Since the side grippers adhere to the wall by simply resting on top of the bolts, KlimBot had to to be

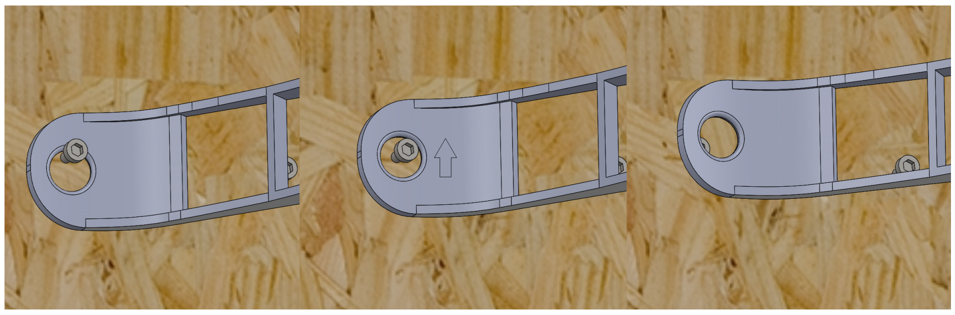


Figure 4.3: Elevating KlimBot in order to release side grippers

elevated slightly before the side grippers could be released, as illustrated in figure 4.3. In order to accomplish this, torque had to be disabled for all four servos on the upper and lower limb before carrying out the same approach as described above. KlimBot was successfully elevated slightly, in order for the side grippers to be released. Upon release, the unsteadiness caused by the rounded tip of the leg part became apparent, as described in section 3.2.2 on page 27. The skewness described in the same section was also discovered at this point, although the horizontal skewness nearly neutralized the vertical skewness.

4.4.4 Elevating KlimBot

Although KlimBot was somewhat unsteady horizontally when the side grippers were released, it was able to drag itself up one row of bolts when supported by hand, by simultaneously retracting the upper limb and stretching out the lower limb. Subsequently regaining adhesion with the side grippers and retracting the lower limb to complete the step up one bolt, was a relatively straightforward process, although KlimBot should have preferably been able to lift itself slightly higher up when fully retracted, in order for both the side grippers and the lower gripper hole to get placed over the bolts without touching the bolt heads. At this point a security measure for preventing self-destruction was introduced. A Processing function was written for constantly reading the present load on the servos during locomotion, in order to be able to shut them off when exceeding a specified threshold value, an incident most likely to occur when failing to release KlimBot's grip before retracting or stretching out. The threshold value was determined empirically.

4.4.5 Steepening the wall

As the tests on a lowered wall was rather successful, the test wall was gradually steepened and KlimBot's gait was tested further. The transition to steeper degrees of the test wall was relatively seamless, until reaching verticality. With a vertical wall, gravity no longer forced KlimBot against the wall, so adhesion now only depended on the grippers hanging onto the bolts. KlimBot remained securely attached to the wall when all four grippers were attached. However, when releasing the upper leg before stretching out, it became obvious that simply placing the side grippers over the bolts would not suffice. KlimBot leaned out from the wall, and could no longer reach back to the wall after stretching out. In order to achieve a more secure adhesion with the side grippers, KlimBot had to be able to retract its horizontal limbs further, reducing the distance between the side grippers and in this way getting a tight grip on the bolts. At this time, the design was modified accordingly, as described in section 3.3 on page 28.

4.4.6 Vertical climb

After improving the design and reassembling KlimBot, the testing proceeded. The thinner leg parts allowed a tighter grip on the bolts with the side grippers, forcing KlimBot closer to the wall. They also made KlimBot able to drag itself slightly higher up when fully retracted, which lead to a cleaner attachment of the side grippers as well as the lower gripper. To further reduce the risk of KlimBot leaning out from the wall when the upper gripper was released, the lower limb pushed the lower part of KlimBot's body further out from the wall, with the effect of pushing the upper part of KlimBot's body closer to the wall. Stability was increased considerably by the widened and less rounded end of the leg part, in combination with better balance as a result of removing skewness, and KlimBot was now able to complete a whole step upwards, on a vertical wall, without being aided by hand. Since positions were given to the servos as actual values, and not as the change in value, the downwards gait could not be directly derived by simply reversing the order of instructions. The reverse sequence of instructions was therefore calculated step by step. Nevertheless, several modifications had to be done to the instructions before succeeding to climb downwards. The precision required for releasing a grip is much higher than what is required for gripping a bolt, since slight contact with the bolt head during release is more likely to cause a hang up.

4.4.7 Horizontal climb

The lead through method was also used for programming the leftwards gait. When both the upper and the lower grippers are released, the side grippers are not sufficient for resisting gravity and holding KlimBot's body parallel to the wall, resulting in the lower limb swinging into the wall. This problem has been dealt with by releasing the lower leg and placing it against the wall as support, prior to release of the upper leg, as seen in figure 4.4 on the next page. In this way, KlimBot was able to slide steadily to the next column of bolts. KlimBot's current most insecure operation is the release of the rearmost gripper after performing sideways locomotion. Making KlimBot able to lift itself slightly higher when performing this release could reduce the risk of getting a hang up with the bolt head.

The rightwards gait was basically achieved by mirroring the instructions for the leftward gait, sending the same positional instructions in the same order, but readdressing all horizontal instructions to the correspond-

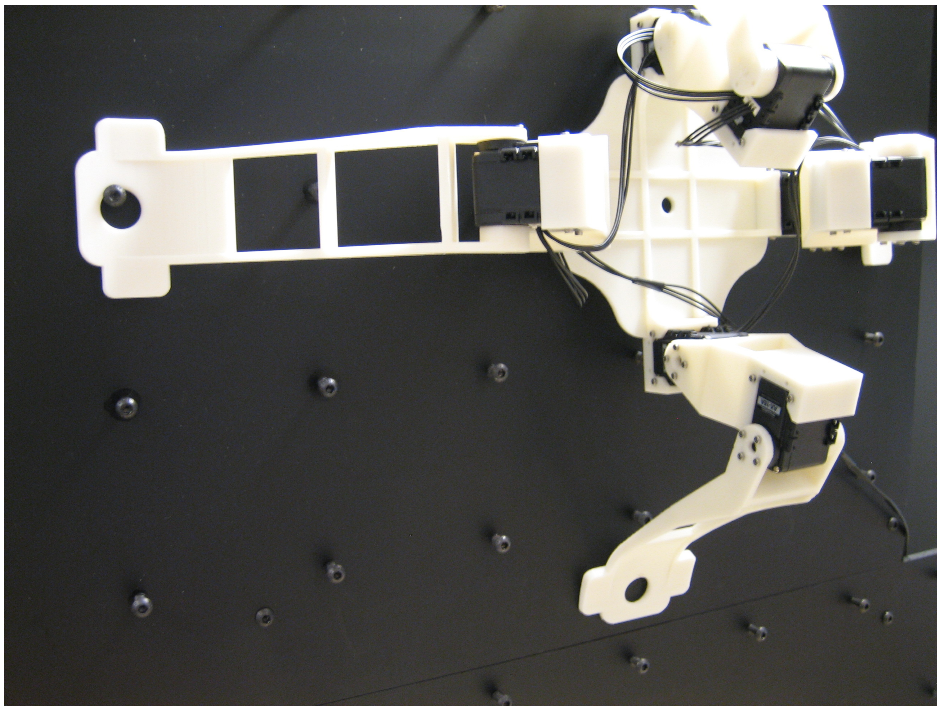


Figure 4.4: KlimBot uses lower leg for support during horizontal climb

ing servos on the opposite side. At this point, KlimBot had successfully moved in all four directions.

4.4.8 Increasing reliability

On some occasions during testing, the upper leg lost the grip of its bolt, resulting in KlimBot loosing its adhesion to the wall. Although KlimBot was usually caught before any damage was inflicted, there was one case were the lower leg snapped as seen in figure 4.5 on the facing page, as it was stuck between the wall and the bolt head whilst the upper part of KlimBot lost grip and fell outwards from the wall. A new leg part had to be printed in order for testing to proceed, and an improved design was made as described in section 3.4.1 on page 29. In addition to the design modifications, some programming modifications were also made, as it was discovered that the order in which instructions were sent to the Dynamixel servos allowed the lower leg to slightly start stretching out before the upper leg started retracting when performing an upwards step, making the upper gripper hole briefly lose contact with its bolt. Additional programming improvements were done for creating more smooth movements. After these measures were introduced, reliability has increased, and KlimBot's upper leg has not since lost its grip to the wall, when travelling at normal speed. However, KlimBot still experiences a few hang ups during sideways locomotion.

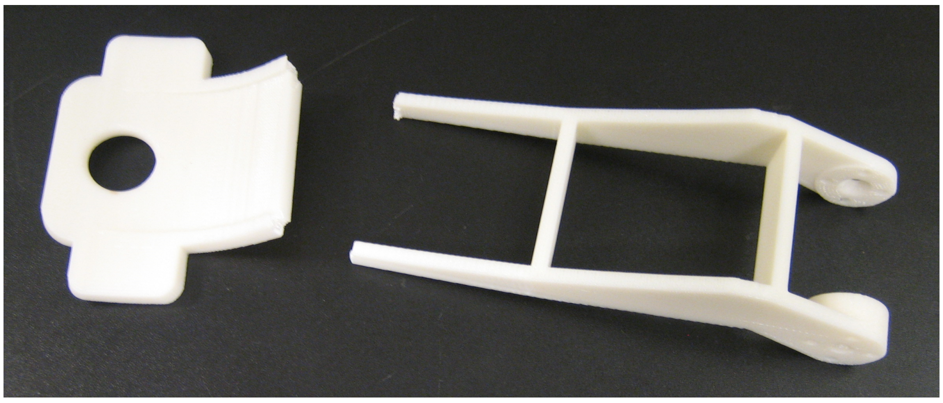


Figure 4.5: Broken lower leg

4.5 Further possible testing

After the testing phase was brought to conclusion, KlimBot was able to successfully climb the vertical wall in all four directions. The servos are very precise, making each execution of KlimBot's gait practically identical. However, KlimBot is very vulnerable to even slight inaccuracies of the wall, such as uneven distances between the bolts, inaccurate alignment or how far the bolts are screwed into the wall. An imperfect climbing gait increases the vulnerability, and KlimBot experienced hang ups on a few of the bolts on the demonstration wall. There are strict requirements regarding reliability for a successful wall-climbing robot, and increasing reliability would be the main priority for further testing.

Firstly, increased reliability could be achieved by creating more smooth movements. Reducing the distance between the positional instructions given to the servos would increase the resolution, giving a smoother motion. As KlimBot is relatively light, coarse and uneven motions could lead to slight jumps, increasing the risk of loosing the grip of a bolt.

Secondly, reliability could be increased by optimizing the climbing gait. With the current gait, the gripper holes were occasionally not perfectly aligned with the bolts, causing slight contact between the bolt head and the gripper during release or adhesion, with the risk of hang ups.

Another possibility for increasing the reliability is to use the left and right leg for support during elevation, in the same way that the lower leg is used as support during sideways locomotion. However, this would reduce climbing speed, and contravene one of the arguments for choosing this specific design.

Tests could also be done in order to increase speed. The current speed is only approximately 15% of the speed offered by the servos used. Increased reliability with respect to adhesion would allow an increase in speed. Increased average speed could also be achieved by increasing the speed for certain sequences, rather than increasing the speed for the complete sequence as a whole.

When performing consecutive sideways steps, a modified gait could further reduce the time spent for traversing a path, by only reattaching the upper gripper between each step, whilst the lower limb maintains its supportive function. By not reattaching the lower gripper, a few seconds could be saved for each consecutive sideways step.

Chapter 5

KlimBot Control System

As the main problem of this thesis suggests, the objective was to make a control system that was easy to use, enabling an operator to easily program the robot to move as desired, possibly over a large number of bolts. With this in mind, the control system has been created as simple as possible, without to many options or alternatives.

5.1 Graphical user interface

5.1.1 Design/layout

For simplicity, the control system has been developed as a Java application for personal computers, using Swing components, as seen in figure 5.1 on the following page. However, the visual design and layout has been composed as if the control system was to be implemented on a designated control device, or as an application for a hand-held device with touch detection. The design seeks to resemble such devices in order to provide a user-friendly and intuitive environment through visual representation. Textual representation has been restricted to a minimum, and well-known symbols are used for the controlling buttons. For visual feedback, the system displays a graphical representation of KlimBot on the bolted wall.

5.1.2 Modes

When running the control system software, the user has been given the possibility of choosing between four different modes, depending on the desired behaviour of KlimBot. The different modes provide different degrees of control regarding path generation, ranging from full user control, letting the user dictate each single step, to full automatization, in which full control of the path planning is handed over to the control system software. The different properties of the four modes are as follows:

- *Manual control.* The default mode is the manual control mode, which lets the operator control KlimBot's locomotion one step at a time. Upon receiving instruction, KlimBot completes one step in the

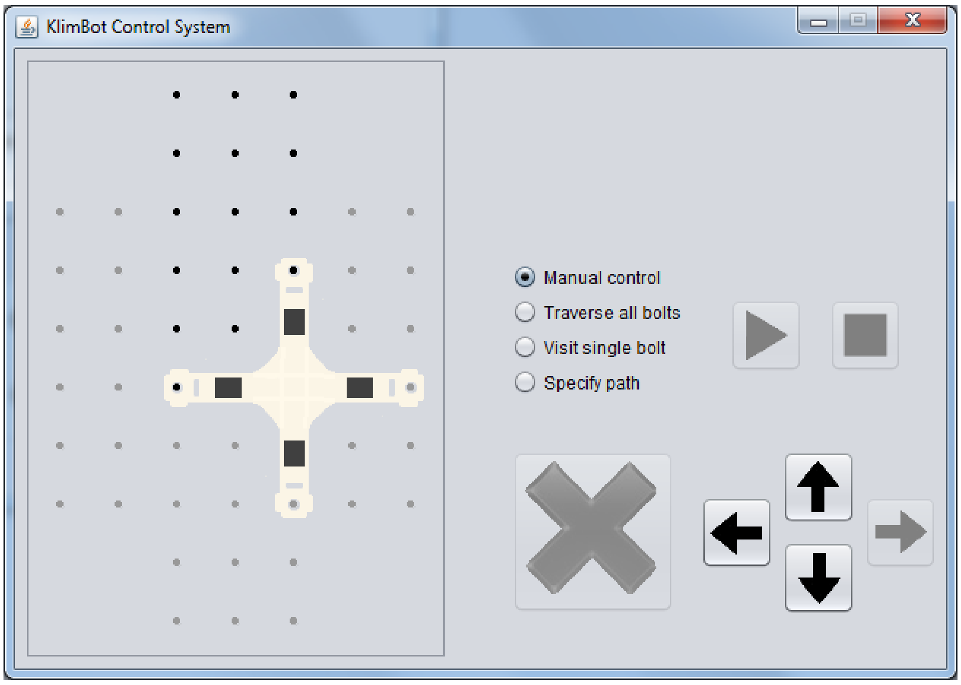


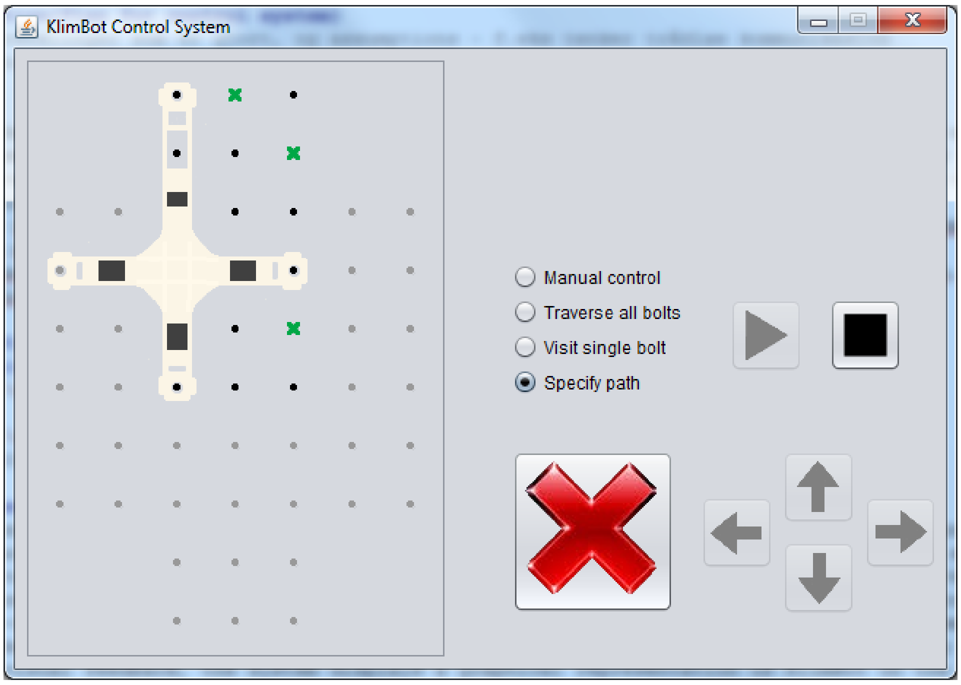
Figure 5.1: KlimBot Control System

specified direction, before returning to an idle state, waiting for a new instruction.

- *Traverse all bolts.* By choosing this mode, the operator lets the software determine a path for full traversal, visiting all reachable bolts at least once. This mode might be beneficial for tasks such as inspection, for guaranteeing that the whole climbable area has been searched and inspected.
- *Visit single bolt.* This mode enables the user to select a single bolt for KlimBot to visit. The path generation is performed by the control system software.
- *Specify path.* When in this mode, KlimBot visits the selected bolts in the order specified, illustrated in figure 5.2 on the next page. The paths between each bolt are generated by the software. This mode is suitable when wanting to visit a set of bolts, or when wanting to explicitly describe each step of a desired path, without having to wait for each step to be performed, as is the case when in the manual control mode.

5.1.3 User interaction

The resemblance to hand-held devices is also reflected by user interaction. All input is given by pressing the mouse button, to simulate touches on a touch screen. The desired mode is set by pressing the corresponding radio



Manual control
Traverse all bolts
Visit single bolt
[✓ Specify path]

Figure 5.2: Specify path mode

buttons. Selection of specific bolts is also done by pressing their graphical representations, selected bolts are labelled by a green cross, as seen in 5.2. KlimBot behaviour is manipulated by pressing the control buttons. For simplicity and intuitiveness, the control buttons are only enabled when some reaction will occur by pressing them.

The control buttons present are:

- *Play.* The play button is used for initiating KlimBot's traversal along a path.
- *Stop.* The stop button is enabled whenever KlimBot is traversing a path. If pressed, KlimBot completes its current step before stopping. Traversal can be resumed by pressing the play button.
- *Abort.* The big red cross is intended for more acute situations, pressing this button leads to an immediate halt.
- *Directional arrows (left, right, up, down).* The directional buttons are used for moving KlimBot when in manual control mode.

5.1.4 Graphical representation

The control system software provides a graphical representation of the actual configuration of the wall of bolts, with the position and state of KlimBot drawn on it. In order to clarify the range of KlimBot, reachable bolts are coloured black, whilst supporting bolts are coloured gray. The graphical representation changes accordingly to the actual robot, so it

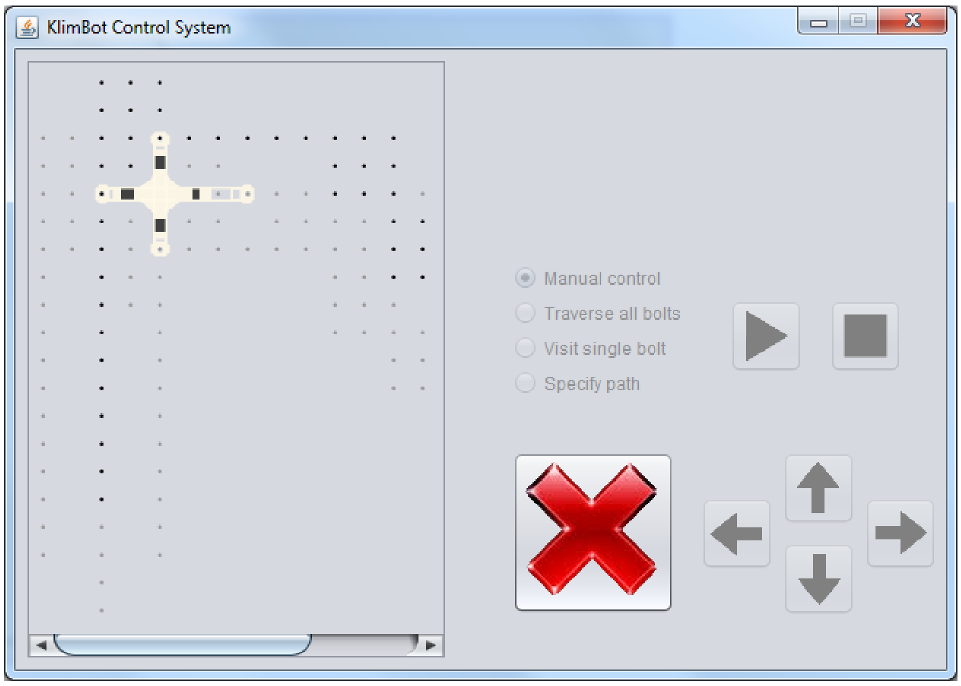


Figure 5.3: Graphical representation of the demonstration wall

provides a means of knowing where KlimBot is positioned, and how far it has reached in its current movement. The thought behind this is to let the operator be in control of the robot without actually having to watch it on the wall before moving it by using the control system. It's also much easier and less tiring to simply move your sight slightly on the screen (from buttons to the graphical representation) instead of having to lift your head of the screen, refocus to see the robot, and refocus again when moving your eyes back to the control screen. Such a graphical representation also facilitates for remote control from a considerable distance, when the climbing robot is out of sight, or operating in hazardous environments.

Figure 5.3 shows a graphical representation of KlimBot outstretched rightwards whilst climbing the demonstration wall.

5.2 Implementation of control system

Processing is Java based, as mentioned, but unfortunately Processing sketches does not support the use of AWT or Swing components. However, all Processing Applets extend the PApplet class, which in turn extends from java.awt.component, meaning the Processing sketch can itself be treated as a Java component. The processing.core.jar library, which contains the basic Processing functionality, including the PApplet class, was imported to the KlimBot Control System NetBeans project. This way, a modified version of the Processing sketch that was described in section 4.2 on page 35 was embedded in the control system as a PApplet component. Since the desired functionality of the control system is operation of KlimBot and not testing,

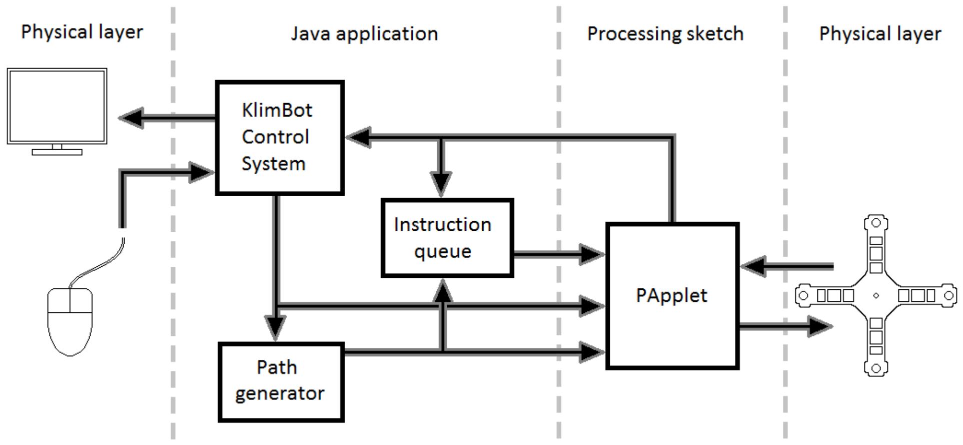


Figure 5.4: KlimBot Control System architecture

this modified sketch excluded the test functions called upon by keyboard input.

5.2.1 Architecture

The high level software architecture of the KlimBot control system is illustrated in figure 5.4. Actions are triggered by user input, given as mouse clicks. When in manual control mode, the control system sends the specified instruction directly to the PApplet by starting a new thread. When in any of the other modes, a path has to be generated, and a specific thread is created and run for these calculations. After a path has been generated, the first directional instruction is sent to the PApplet as a new thread by the path generator, before the remaining instructions are stored in an instruction queue. The PApplet is responsible for all communication with KlimBot. After receiving an instruction, the PApplet sends several commands to the servos and constantly receives feedback regarding the servos positions and load. Status updates are sent from the PApplet to the control system, which in turn presents the user with visual output. Whenever an instruction is completed, the instruction queue is checked. If there are remaining instructions in the queue, a new instruction thread is created and run. By threading the time consuming tasks, particularly the communication with KlimBot, the control system is able to produce up to date graphical output, meaning the user input leads to instant graphical response, without having to wait until KlimBot has completed its movements.

5.2.2 Path generation

When in any other mode than the manual control mode, a climbing path has to be generated for KlimBot. The bolted wall can be viewed as an unweighted and undirected graph, often referred to as a simple graph. The vertices of the graph represent the bolts, and the edges between them

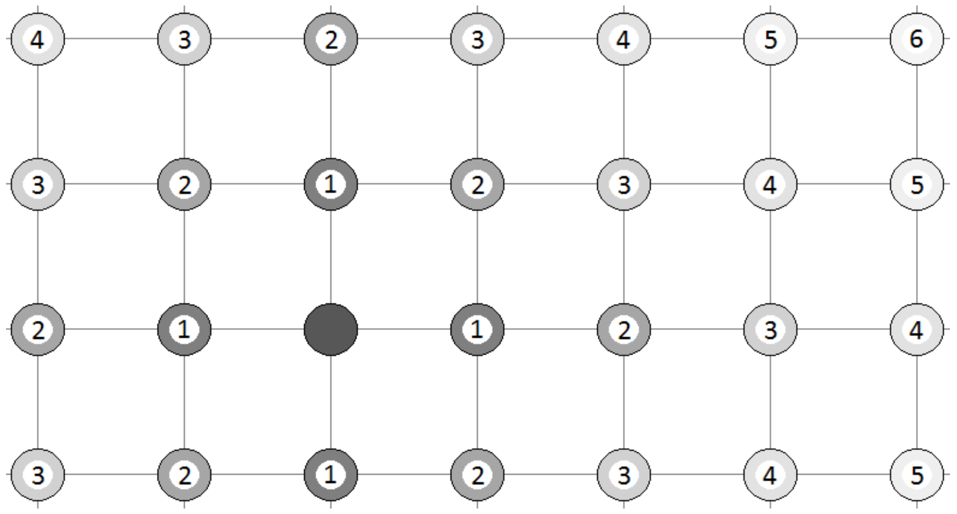


Figure 5.5: Breadth-first search expands by one edge on every iteration

represent a step to the neighbouring bolts, vertically and horizontally, meaning any vertex can have at most four edges. The search algorithm implemented for path generation in the KlimBot control system is breadth-first search.

Breadth-first search is a well known search strategy for unweighted graphs. The search starts at a root vertex (KlimBot's current position) and explores all the neighbouring vertices. For each of those vertices, it explores any neighbouring vertex that hasn't already previously been explored. The search expands in this fashion until the goal vertex is found. Breadth-first search expands the search by one edge on every iteration, with the effect of finding the the smallest number of steps it takes to get to any given goal vertex from the root vertex. The graph traversal of breadth-first search is illustrated in figure 5.5.

Breadth-first search is complete, meaning it will always find a solution whenever one exists. Furthermore, the algorithm guarantees an optimal solution when searching for a path between two specified vertices, as is the case for the "visit single bolt" mode. Optimality is also guaranteed for the "specify path" mode, as the search problem can be viewed as searching for subpaths between the vertices in the order that has been specified by the user. However, optimality is not guaranteed when wishing to traverse all bolts. Nevertheless, breadth-first search provides a satisfactory solution at a reasonable speed.

By big O notation, the time complexity of breadth-first search can be expressed as $O(|V|+|E|)$[19], since every vertex and every edge will be explored in the worst case.

The graph representation of the bolted wall is treated as unweighted, since there is an equal distance of 100mm between all neighbouring bolts. However, for the current configuration of KlimBot, the requirement of using the lower leg for support during sideways locomotion, means sideways locomotion is slightly slower (has higher cost) than moving

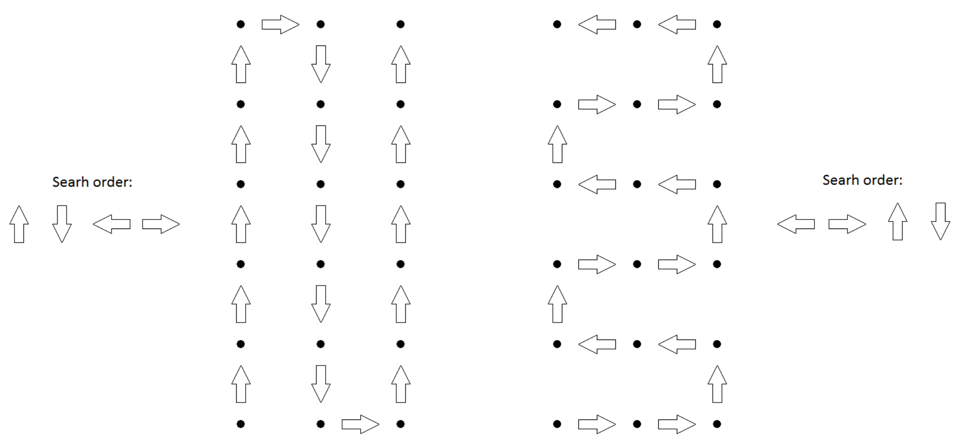


Figure 5.6: Comparison of search order

upwards or downwards. To minimize the effect of this cost gap, the order in which neighbouring vertices are explored is configured to be *up - down - left - right*, hence favouring vertical movement. The effect of different search orders is illustrated in figure 5.6. Each vertical step has been measured to take approximately 10 seconds, whereas the horizontal steps were measured to approximately 13.5 seconds. The traversal on the left side of figure 5.6 is completed approximately 20% quicker than the traversal on the right side of the figure.

If the graph was not considered unweigthed, Dijkstra's algorithm[19] could be implemented. A heuristic search method such as A* search[15] could also be considered, for improving search time performance. An other alternative, with lower space complexity, is the iterative deepening depth-first search algorithm.

5.3 Further possible improvements

A few possible improvements on the control system are suggested in the following text.

The control system should offer the ability to save paths for later use, whether the paths are created by manual control or automatically by the control system. Combining saved paths to make longer paths is also an option.

A replay button could be created, facilitating continuous traversal of a specified path. Other possibilities are options for specifying the number of traversals, or at what time intervals traversal should be performed.

For the "specify path" mode, the green crosses specifying the path could be replaced by numbers describing the order in which the bolts would be visited, for increased overview and readability.

The control system could also offer the ability to modify the speed of KlimBot's servos.

Especially for bigger walls, the possibility of zooming in or out on the graphical representation could be beneficial. Zooming out would provide a better overview, whilst zooming in could give the level of detail required for specifying particular bolts.

When the abort button is pressed, or when the specified load threshold has been exceeded, the servos are shut down by disabling their torque. With the current implementation, the program has to be restarted in order for KlimBot to return to normal operation. Some sort of recovery could be implemented, for example for reversing the last instructions and make KlimBot return to the last bolt, in order to do another attempt at the same instruction, or to choose another path.

Part III

Discussion

Chapter 6

Discussion

In order to determine whether or not the results of the research on KlimBot and its control system provides a satisfactory answer to the main problem, the criteria for success have to be discussed. KlimBot is able to climb a vertical, bolted wall, and can be quite easily controlled with the KlimBot Control System. However, KlimBot depends on quite a few assumptions and simplifications for working properly, thus making it less applicable for an environment in which a commercially manufactured climbing robot might be expected to operate. The following discussion attempts to form a foundation for answering the main question in such a context.

Firstly, the research on KlimBot is thoroughly discussed, considering the design, the test approach and test process as well as the control system implementation. Thereafter, a discussion follows on the requirements and challenges present for enabling the transition to a real-world environment.

6.1 KlimBot discussion

6.1.1 Design discussion

To avoid confusion, design A, the design concept leading to the actual design of KlimBot, will be referred to as KlimBot in this section, and the alternative design will be referred to as design B.

As mentioned in section 3.1.3 on page 23, the design choice might have been different if it was made at a later point of the project. During work with KlimBot, a few challenges were revealed that design B might have been more suitable for dealing with. Additionally, after the idea for design B has had time to mature, an improved climbing gait has been discovered, and attention has also been drawn to a few features solely offered by design B that might have proven useful.

KlimBot is very vulnerable to even slight irregularities when it comes to how far the bolts are screwed into the wall, compared to the bolts used when KlimBot's gait has been programmed. If the bolts extend too far out from the wall, the grippers will get stuck to the bolt head when trying to release its grip. On the other hand, if the bolts are screwed to close to the wall, KlimBot may try to grip the bolt before having completely passed the

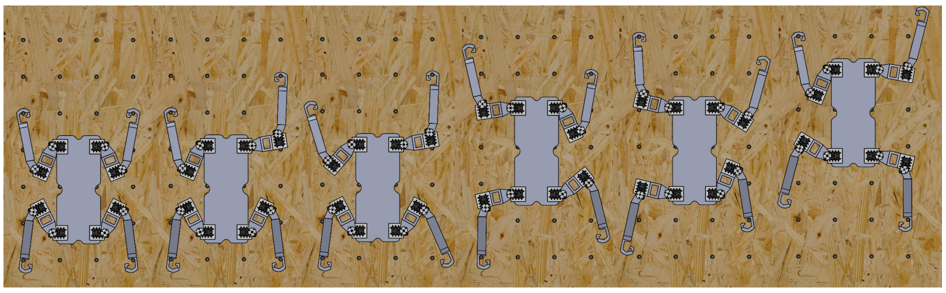


Figure 6.1: Optimized climbing gait for design B

bolt head, thus failing to adhere. As the gripper of design B slides onto the bolts from the side, the bolt heads would not pose a threat, in fact, increasing the diameter of the bolt heads might actually reduce the risk of loosing grip and falling outwards from the wall.

Furthermore, even with the improved leg design, KlimBot is somewhat unsteady during vertical locomotion. Although the remaining limbs could be used as support for increased steadiness by being placed onto the wall, no more than two grippers can be attached to the wall whilst moving from one bolt to another. Design B provides a higher degree of flexibility. If required, all four grippers could be attached to bolts during locomotion, thus increasing the adhesion reliability. However, design B is also able to climb with only two of the grippers attached to the bolts. The optimized climbing gait for design B, as illustrated in figure 6.1, might contravene the argument claiming KlimBot has the highest velocity potential. In fact, when climbing with this gait, each leg only has to grip onto bolts on every other row, effectively doubling the potential for climbing speed. Moreover, the two limbs that are released during locomotion serve as support because they are always in contact with the wall, making design B more steady than KlimBot, even when only two grippers are attached during locomotion.

Having all four grippers attached during locomotion consequently reduces the required torque for each servo, as well as reducing the stress applied. This in turn, facilitates for carrying heavier payloads.

With increased reliability, the servos could also be set to higher speed. The flexibility offered by design B also facilitate research regarding the trade off between speed and reliability. The climbing velocity is determined by two factors, the servo speed and the climbing gait. Recalling that KlimBot's servos were only running at approximately 15% of maximum speed, there is a great potential for increasing speed. The reliability provided by having all four grippers attached during locomotion might allow considerable increase in servo speed. Might this increase in servo speed even be able to exceed the velocity increase allowed by the optimized climbing gait? Further research on this subject would be very interesting.

Design B also has the potential of diagonal locomotion, which could drastically increase traversal speed. With a few design modifications, such as slightly increasing the length of the thigh part, design B would be able to stretch out far enough to reach the bolts diagonally. The flexibility of design B's gripper position on the wall also offers a potential of overcoming inaccuracies in the bolted wall, whereas KlimBot depends on a near-perfect alignment of the bolts, both vertically and horizontally, even with an optimized climbing gait.

Furthermore, the ability of moving the body base around while all four grippers are still attached gives the possibility of bringing the body to essentially any desired position, which for instance could be beneficial when carrying operational tools that are meant to perform tasks at specific positions on the wall. KlimBot, on the other hand, can only move its body linearly, corresponding to the grid made up by the bolts.

However, the grass is always greener on the other side, and further practical work with design B would most likely reveal new problems that have not been considered and discussed. Still, design B undoubtedly possess characteristics that outperform KlimBot for certain tasks.

Nonetheless, there are challenges and limitations present for both designs that would have to be overcome in order to create a final and fully functional wall-climbing robot for bolted walls. Creating a hybrid version of the designs for combining the best of the two worlds could be an advisable approach for further research. By adding servos for creating extra joints between the body base and the limbs of design B, with rotational axes parallel to the wall, the qualities previously reserved for KlimBot would be gained, such as the ability of elevating limbs or body part out from the wall, thus also being able to climb bent surfaces.

6.1.2 Test discussion

As described in section 4.3 on page 35, the choice of test approach for determining the appropriate position values to send to the servos, was mainly based on a desire to get KlimBot to the wall as quickly as possible, in order for the test process to commence. In addition, the objective was to define a sequence of positional commands combined to form a complete climbing gait that an operator could simply replay directly, rather than providing the possibility of making small manoeuvres or adjustments. Initially, the use of inverse kinematics was intended for calculating the desired position values directly, which seemed to be an unnecessarily complex and advanced approach compared to the preferred option, the lead through method. However, the eagerness to get going with the test phase might have made the actual test process less efficient and rewarding than it could have been. Although it was not the final objective to equip an operator with a teach pendant, such an instrument could have been of great value during programming of the climbing gait.

By creating a designated teach program with the implementation of inverse kinematics, a teach pendant could be expressed by a computer keyboard or even a PlayStation Controller. Small adjustments of the coordinates of KlimBot's gripper hole for a specified leg could be given as controller input to the teach program, which in turn would calculate the required joint angles by applying inverse kinematics, convert these joint angles to the corresponding position values, and send instructions to the servos directly during runtime, as well as saving all position values for the climbing gait along the way.

Both the creation of the teach program itself, and the programming of the climbing gait with the use of the teach pendant, would be a lengthy processes. However, it could still prove to be more time efficient in the long run. The high resolution provided by the small incremental adjustments in position value help produce a very precise and smooth climbing gait. The possibility of having torque enabled for all servos during the programming means the behaviour of the servos would bear greater resemblance to actual operation behaviour compared to that achieved by using the lead through method, where the torque for the servos has been disabled. The combination of these factors could drastically reduce the amount of error correction needed, thus reducing the total time spent. Although the lead through method provided a fairly straightforward and effective way of getting started with the test process, a lot of time has been used on error correction and improvements of KlimBot's climbing gait. Despite these efforts, there are still a lot of improvements that should be done in order to increase reliability. An optimized climbing gait would most likely contribute to reducing the risk of hang ups.

As mentioned earlier, the increase in reliability, combined with smoother motion, would also provide a greater potential for increasing servo speed, something that could be of importance when traversing greater distances on bigger walls.

In the current implementation, positional instructions are given to the servos as actual values. It might have been a better idea to send the instructions as relative positions instead, as this could make it easier to perform slight adjustments on the gait as a whole. There should also be implemented some functionality to ease the reversal of instructional sequences, for instance by saving all servo instructions in an appropriate data structure. Rather than simply running through sequential lines of instructional code, the order of instructions could then be easily reversed by switching the direction of iteration over the instructional collection. With this functionality available, it could be easier to apply the capability of returning to the start position of the current directional step, in the event of a hang up or if the abort button is pressed. No such functionality for recovery is present with the current implementation.

The transition from the test wall to the larger demonstration wall gives some valuable experience which can help when trying to answer the main problem of this thesis. The size and configuration of the demonstration wall bears a greater resemblance to a realistic environment in which a wall-climbing robot might be applied. The impression developed through the test process suggests that programming a gait used for climbing a few bolts is easier than programming a gait for traversal of several bolts, even though the bolt configuration is supposed to be identical. The testing indicates that expansion of the wall, with an increase in the amount of bolts, lead to a higher probability of configuration inaccuracies sufficient to cause trouble for KlimBot. The heightened risk of missteps following an increase in the amount of bolts is easily comprehensible when having in mind that only a few bolts are used as reference when programming the climbing gait, as well as considering the strict requirements for precision in order for the gripper hole of KlimBot's leg to successfully adhere to the bolts.

6.1.3 KlimBot Control System discussion

KlimBot's current location on the wall is emulated by the graphical representation of the control system. However, the original location of KlimBot, serving as the base for all further locational calculations, has to be given explicitly to the software at start up, and a correct graphical representation as well as safe behaviour depends on that stated information to be true. The safety measures implemented to prevent KlimBot from climbing out of bounds are determined by the control systems internal perception of current location, in combination with the provided information for describing the configuration of the bolts on the wall. If erroneous information is provided, the control system could allow KlimBot to follow a complete sequence of positional instructions, even though no reachable bolts are present, resulting in an attempt to adhere to imaginary bolts with the inevitable consequence of falling off the wall. Additional sensory information would have to be provided for the control system to be able to verify KlimBot's actual location, in order for satisfactory reliability to be guaranteed.

Displaying a correct graphical representation of KlimBot's actual behaviour is more certain, because of the continuous positional feedback received by the servos. The servos positional sensors are also the basis for being able to properly determine KlimBot's relative location on the wall, with respect to the alleged original location.

With the current implementation of the control system, path generation is performed during runtime. When all bolts are known, which is the case for both the test wall and the demonstration wall, all optimal paths could actually be calculated in advance, and saved in a lookup table for easy access when needed. However, by big O notation, the space requirement for such a lookup table becomes $O(|V|^2)$, since the optimal paths to all vertices have to be stored for every vertex. Hence, in practice, if the number of bolts would ever make search time an issue, both the excessive time required for calculation all optimal paths and the space requirements of the lookup table approach would make it inapplicable. For challenges regarding complexity, a better approach would be to use some sort of informed search strategy depending on the resource limitations, such as A* search[15], or a memory bounded search variant such as Iterative Deepening A* search[18] or Simplified Memory Bounded A* search[38], which might sacrifice the optimality property for staying within the memory boundaries.

A test person with no prior experience with robot control was kind enough to assist in an experiment with the KlimBot Control System as a way of determining whether or not the declared objective of creating a control system which makes it easy for an operator to program the traversal of numerous bolts has been met. Without any guidance, the test person was able to successfully control KlimBot and utilize all the functionalities available.

Although the limited functionality of the control system naturally makes it less complicated to use than what would be the case for a control system of a more a complex wall-climbing robot, the outcome of the experiment indicates that there is a potential for creating a comprehensible control system by focusing on simplicity and an intuitive and user-friendly layout and design.

6.2 Real-world discussion

In order for the commercialization of a wall-climbing robot to be feasible, with the approach of gripping onto bolts, some tough challenges would have to be conquered. The challenges present depends on the expected behaviour, as well as the environment in which the robot is supposed to operate. This section contains a discussion on the transition challenges from a prototype to an applicable wall-climbing robot.

6.2.1 Advanced grippers

The adhesion concept of gripping onto bolts on the wall shows great promise for acquiring a secure grip. However, a general problem with any of the designs proposed is the reliability of the grippers. Adhesion by simply placing a gripper over the bolt depends on the bolts to counteract the force exerted on the climbing robot, which works fine when gravity is the main force, pulling the grippers towards the bolts. However, since the grippers have no contact with the underside of the bolt, the robot is left vulnerable to the exertion of upwards forces, and could easily lose grip when such forces are applied. A more advanced gripper should be able to grab onto the bolts in such a way that external forces from all directions can be cancelled out by contact with the bolts. An idea for such a gripper is a claw or clamp inspired by the chuck of a drill, such as the one shown in figure 6.2 on the facing page. A specialized claw could lock onto the bolt by tightening its claw onto the bolt as well as encapsulating the bolt head, ensuring a secure grip, even under the influence of forces directed outwards with respect to the wall. Additionally, the ability of adjusting the hole size increases the tolerance for slight inaccuracies in the configuration of bolts on the wall. With a more secure adhesion, the climbing robot could be imagined to climb overhanging walls, or even roofs. It would also tolerate much more external disturbance and vibrations, for instance caused by bad weather.

KlimBot's most error-prone operation is the release and attachment of its grippers. Improved grippers could increase reliability in two ways, both by providing a more secure grip once attached, as well as facilitating safer release and attach operations.

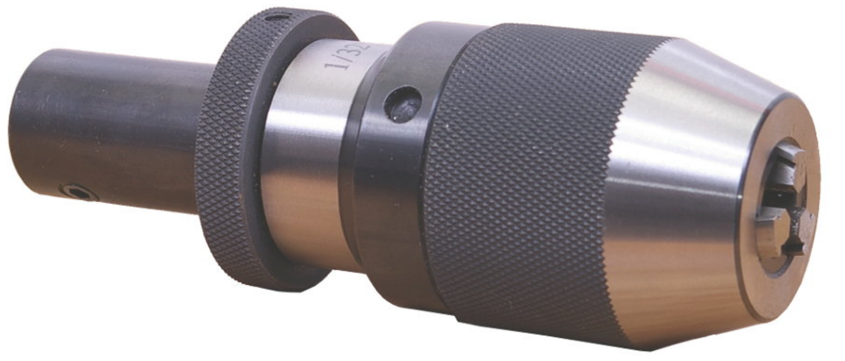


Figure 6.2: Drill chuck[40]

6.2.2 Artificial intelligence

In order for an operator to easily control a climbing robot, a high degree of autonomy would have to be implemented, especially when considering the increase of functionality and complexity required for a commercial wall-climbing robot. The demand for autonomy implies the need of implementing some sort of artificial intelligence. In the field of artificial intelligence, an intelligent agent is an entity that perceives its environment, before reasoning based on the acquired information, and acting in the way it considers to be most appropriate in order to reach some defined goal[39]. The absence of any advanced sensory equipment limits KlimBot's capability of perceiving the environment it operates in. The only sensory information acquired is the information received from the internal sensors of the servos. KlimBot simply replays a preprogrammed sequence of instructions for moving in the desired direction, and hopes this is sufficient for a successful move. To gather the information needed to develop a sufficient understanding of its surroundings, in order to deduce appropriate behaviour, as well as to meet the requirements for reliable operation, a commercial climbing robot would require additional sensory devices. Cameras are needed for locating bolts, and tactile sensors to ensure secure adhesion is acquired before locomotion is initialized. Furthermore, extra sensors would have to be present for guaranteeing the correct operation of any tools. The approach of having one specific climbing gait which is repeated for every step would prove to be insufficient, and artificial intelligence would have to be implemented for the robot to be able to determine an appropriate climbing gait during runtime.

6.2.3 Power supply

Depending on the size of the wall the climbing robot is supposed to climb, receiving electrical power through a cable would most likely become impractical if not infeasible. A potential solution could be to have some sort of rechargeable power supply mounted locally on the robot, and provide a number of selected bolts with the functionality to act as charging docks.

One of the essential advantages with the adhesion method of gripping onto bolts, is the concept of passive adhesion. In the event of a power cut, the robot would still be able to maintain a secure grip, providing superior reliability compared to some of the other adhesion methods, such as those dependent on fans for creating negative pressure.

6.2.4 Wall prerequisites

Some of the major disadvantages for the approach of climbing bolted walls, are the strict prerequisites regarding the walls on which the robot can operate. Highly specialized and designated walls are required, with specific bolt configurations corresponding to the implementation of the climbing robot. In order for a commercially manufactured climbing robot for bolted walls to be feasible and realistic, these prerequisites would have to be made less strict.

Firstly, the density of bolts would have to be reduced compared to that of the walls created for KlimBot, where every fully bolted square metre consists of 100 bolts. Scaling up the size of the robot could be one possible suggestion, consequently increasing the weight of the robot. Further reduction of the required density of bolts could be achieved by applying prismatic (linear) joints for facilitating adaptive limbs with an increased reach length. Adaptivity would also provide more flexibility when planning the appropriate climbing gait.

Secondly, the requirements concerning bolt configuration could be made less specific. The requirement for a a perfectly aligned grid of bolts might be too rigorous, especially if the required precision is as high as set by KlimBot, where even slight inaccuracies would result in missteps and malfunction. Although a more intelligent and flexible climbing robot would manage to neutralize such inaccuracies, the requirement for having the bolts configured as a grid might not necessarily be the most practical solution, as it makes the construction of the wall rather demanding. A climbing robot that is able to plan an appropriate climbing gait for reaching each separate bolt, could manage with a less specific configuration. The bolts might even be placed randomly, as long as a certain density is maintained as well as the distances between the bolts are kept within some specified range.

For increasing the applicability of the climbing robot further, it should not be limited to only operate on flat walls. Nuclear storage tanks, oil rigs and other constructions for which a climbing robot might be considered suitable, consist of more advanced structures with bent walls, overhangs and various irregularities or obstacles. Advanced grippers and sufficient degrees of freedom, in combination with artificial intelligence and advanced computational logic, would be the minimum requirements for dealing with such challenges.

When generalizing and simplifying the specified requirements for the wall, the need for a correspondingly advanced climbing robot is implied. Both the hardware and software of the robot would have to be upgraded, in order to deal with the increased complexity of tasks such as navigation, localization, gait generation, mapping and path planning.

6.2.5 Optimization

Evolutionary computational techniques, such as those described in section 2.4.1 on page 8, could be used for optimizing both the physical design and the locomotion of a climbing robot[29]. Variables such as the length and size of the different robot parts could be optimized in such a fashion, as well as the number of joints or legs. Optimization of design and climbing gait could increase the robots maximum step length, as well as making the robot more efficient with regards locomotion velocity and power consumption.

Additionally, the discipline of machine learning[2] could optimize the behaviour of a climbing robot. Machine learning algorithms operate by interpreting input in the context of experience and makes decisions and predictions based on these, rather than simply following some preprogrammed instructions.

6.2.6 Additional challenges

As mentioned, the challenges present depends on the expected behaviour, as well as the environment in which the robot is supposed to operate.

The performance of certain tasks might require heavy or big tools, and the design of the climbing robot would have to be modified accordingly. A heavy payload would also require an increased torque to be delivered by the actuators, consequently increasing the power consumption. The actual operation of the tools might also require complex software. Furthermore, a heavy payload would also increase the required strength of the bolts to be applied.

The appropriate building material used for the climbing robot might also need certain properties for accommodating to the surrounding environmental challenges. Rough weather, saline sea water, extreme temperatures and radiation are only some of the potential hazards. The materials applied must possess the required strength and tolerance, and vulnerable components might have to be encapsulated.

Chapter 7

Conclusion

The stated main problem of this master's thesis questions the feasibility of creating a reliable wall-climbing robot for bolted walls, as well as implementing a control system for easily controlling the robot's traversal over long distances.

During this research project, KlimBot, a legged wall-climbing robot with the ability of climbing bolted walls, has been designed, built and programmed. Two bolted walls have also been designed and built in order for the climbing robot to be tested. KlimBot has been able to successfully climb the designated walls horizontally and vertically. In addition, a control system has been implemented in order for an operator to easily control and generate the paths for the KlimBot to follow.

In order for this to be achieved, strict prerequisites were set with regards to the climbing wall, the bolt configuration and the work environment. Power supply and control system could be connected to KlimBot by wire, and climbing was the sole functionality desired.

In such a context the project has been a success, providing a positive answer to the main question.

However, in order for the commercialization of such a wall-climbing robot to be feasible, the same assumptions would not hold, and several tough challenges would become present, such as the ones described in section 6.2 on page 58.

Nevertheless, the encouraging results of KlimBot suggests there is a great promise for the approach of climbing a bolted wall. This adhesion method offers a great potential for payload capacity and high reliability, even in the event of a power cut, making such a climbing robot appropriate for performing heavy tasks or for operating in hazardous environments.

In order to get closer to determining the feasibility of the proposed approach, further research should focus on increased flexibility and autonomy.

Bibliography

[1] ABB. *RobotStudio*. URL: http://new.abb.com/products/robotics/robotstudio.

[2] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2004.

[3] British Automation & Robot Association. *ROBOT PROGRAMMING METHODS*. URL: http://www.bara.org.uk/robots/robot-programming-methods.html.

[4] C. Balaguer et al. 'A climbing autonomous robot for inspection applications in 3D complex environments'. In: *Robotica* 18 (03 May 2000), pp. 287–297. ISSN: 1469-8668. URL: http://journals.cambridge.org/article_S0263574799002258.

[5] J.M. Benedetto. 'Economy-class ion-defying ICs in orbit'. In: *Spectrum, IEEE* 35.3 (Mar. 1998), pp. 36–41. ISSN: 0018-9235. DOI: 10.1109/MSPEC.1998.663756.

[6] Geoffrey Biggs and Bruce MacDonald. 'A survey of robot programming systems'. In: *Proceedings of the Australasian conference on robotics and automation*. 2003, pp. 1–3.

[7] L. Briones, P. Bustamante and M.A. Serna. 'Wall-climbing robot for inspection in nuclear power plants'. In: *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. May 1994, 1409–1414 vol.2. DOI: 10.1109/ROBOT.1994.351292.

[8] K. Berns C. Hillenbrand D. Schmidt. 'CROMSCI: development of a climbing robot with negative pressure adhesion for inspections'. In: *Industrial Robot: An International Journal* 35.3 (2008), pp. 228–237. DOI: 10.1108/01439910810868552.

[9] Luke Chilson. *The Difference Between ABS and PLA for 3D Printing*. URL: http://www.protoparadigm.com/news-updates/the-difference-between-abs-and-pla-for-3d-printing/.

[10] Wei-Ging Liu Chin-Pao Hung. 'Intuitive Embedded Teaching System Design for Multi-Jointed Robots'. In: *Int J Adv Robot Syst* 9 (2012). DOI: 10.5772/46126. URL: http://www.intechopen.com/books/international_journal_of_advanced_robotic_systems/intuitive-embedded-teaching-system-design-for-multi-jointed-robots.

[11] Baeksuk Chu et al. 'A survey of climbing robots: Locomotion and adhesion'. English. In: *International Journal of Precision Engineering and Manufacturing* 11.4 (2010), pp. 633–647. ISSN: 1229-8557. DOI: 10.1007/s12541-010-0075-3. URL: http://dx.doi.org/10.1007/s12541-010-0075-3.

[12] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*. springer, 2003.

[13] J. Estremera, E. Garcia and P. Gonzalez de Santos. 'A Multi-Modal and Collaborative Human–Machine Interface for a Walking Robot'. English. In: *Journal of Intelligent and Robotic Systems* 35.4 (2002), pp. 397–425. ISSN: 0921-0296. DOI: 10.1023/A:1022303009950. URL: http://dx.doi.org/10.1023/A:1022303009950.

[14] Jon Excell and Stuart Nathan. *The rise of additive manufacturing*. URL: http://www.theengineer.co.uk/in-depth/the-big-story/the-rise-of-additive-manufacturing/1002560.article.

[15] Peter E Hart, Nils J Nilsson and Bertram Raphael. 'A formal basis for the heuristic determination of minimum cost paths'. In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2 (1968), pp. 100–107.

[16] John H Holland. 'Genetic algorithms'. In: *Scientific american* 267.1 (1992), pp. 66–72.

[17] Chua Chee Kai, Leong Kah Fai and Lim Chu-Sing. *Rapid Prototyping: Principles and Applications in Manufacturing*. 2nd ed. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2003. ISBN: 9812381201.

[18] Richard E Korf. 'Depth-first iterative-deepening: An optimal admissible tree search'. In: *Artificial intelligence* 27.1 (1985), pp. 97–109.

[19] Charles E Leiserson et al. *Introduction to algorithms*. MIT press, 2001.

[20] C. Menon, M. Murphy and M. Sitti. 'Gecko Inspired Surface Climbing Robots'. In: *Robotics and Biomimetics, 2004. ROBIO 2004. IEEE International Conference on*. Aug. 2004, pp. 431–436. DOI: 10.1109/ROBIO.2004.1521817.

[21] Dan Mishek. *How and When to Choose Between Additive and Subtractive Prototyping*. URL: http://www.vistatek.com/pdfs/Choosing-Between-Additive-and-Subtractive-Prototyping-manufacturing.pdf.

[22] S. Mitsi et al. 'Off-line programming of an industrial robot for manufacturing'. English. In: *The International Journal of Advanced Manufacturing Technology* 26.3 (2005), pp. 262–267. ISSN: 0268-3768. DOI: 10.1007/s00170-003-1728-5. URL: http://dx.doi.org/10.1007/s00170-003-1728-5.

[23] Akbar F Moghaddam et al. 'Novel Mobile Climbing Robot Agent for Offshore Platforms'. In: *Proceedings of World Academy of Science, Engineering and Technology*. 68. World Academy of Science, Engineering and Technology. 2012.

[24] m...@paus.ch. *SimpleDynamixel - A simple library to access the robotis dynamixel servo motors for processing*. URL: https://code.google.com/p/simple-dynamixel/.

[25] M.P. Murphy and M. Sitti. 'Waalbot: An Agile Small-Scale Wall-Climbing Robot Utilizing Dry Elastomer Adhesives'. In: *Mechatronics, IEEE/ASME Transactions on* 12.3 (June 2007), pp. 330–338. ISSN: 1083-4435. DOI: 10.1109/TMECH.2007.897277.

[26] Keiji Nagatani, D. Endo and Kazuya Yoshida. 'Improvement of the Odometry Accuracy of a Crawler Vehicle with Consideration of Slippage'. In: *Robotics and Automation, 2007 IEEE International Conference on*. Apr. 2007, pp. 2752–2757. DOI: 10.1109/ROBOT.2007.363881.

[27] Joseph Needham. *Science and Civilisation in China: Volume 2, History of Scientific Thought*. Cambridge University Press, 1991. ISBN: 0-521-05800-7.

[28] Shimon Y. Nof. *Handbook of Industrial Robotics (2nd Edition ed.)* John Wiley & Sons, 1999, pp. 3–5. ISBN: 0-471-17783-0.

[29] Tonnes Frostad Nygaard. 'Evolutionary optimization of robot morphology and control'. MA thesis. University of Oslo: Department of Informatics, 2014.

[30] S. Ohtsuki. *Robot for a work on a wall surface*. US Patent 4,993,913. Feb. 1991. URL: http://www.google.com/patents/US4993913.

[31] M. Osswald and F. Iida. 'A climbing robot based on Hot Melt Adhesion'. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. Sept. 2011, pp. 5107–5112. DOI: 10.1109/IROS.2011.6094741.

[32] M. Perrollaz et al. 'Teachless teach-repeat: Toward vision-based programming of industrial robots'. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. May 2012, pp. 409–414. DOI: 10.1109/ICRA.2012.6224639.

[33] Oxford University Press. *"robotics" - Oxford Dictionaries*. URL: http://www.oxforddictionaries.com/definition/english/robotics.

[34] W.R. Provancher, S.I. Jensen-Segal and M.A. Fehlberg. 'ROCR: An Energy-Efficient Dynamic Wall-Climbing Robot'. In: *Mechatronics, IEEE/ASME Transactions on* 16.5 (Oct. 2011), pp. 897–906. ISSN: 1083-4435. DOI: 10.1109/TMECH.2010.2053379.

[35] ROBOTIS. *ROBOTIS e-Manual v1.23.00*. URL: http://support.robotis.com/en/product/dynamixel/ax_series/ax-18f.htm.

[36] ROBOTIS. *USB2Dynamixel*. URL: http://support.robotis.com/en/product/auxdevice/interface/usb2dxl_manual.htm.

[37] LE Roscoe et al. 'Stereolithography interface specification'. In: *America-3D Systems Inc* (1988).

[38] Stuart Russell. 'Efficient Memory-Bounded Search Methods'. In: *In ECAI-92*. Wiley, 1992, pp. 1–5.

[39] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2009.

[40] shopsmith. *Keyless chuck*. URL: http://www.shopsmith.com/ownersite/catalog/images/keyless_chuck_l.jpg.

[41] Roland Siegwart, Illah Reza Nourbakhsh and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[42] Manuel F Silva and JA Tenreiro Machado. 'New technologies for climbing robots adhesion to surfaces'. In: *Proc. of the International Workshop on New Trends in Science and Technology*. 2008.

[43] Avishai Sintov, Tomer Avramovich and Amir Shapiro. 'Design and motion planning of an autonomous climbing robot with claws'. In: *Robotics and Autonomous Systems* 59.11 (2011), pp. 1008–1019. ISSN: 0921-8890. DOI: http://dx.doi.org/10.1016/j.robot.2011.06.003.

[44] SMG3D. *Insight 3D Printing Software*. URL: http://www.smg3d.co.uk/3d_scanners/insight_software.

[45] Mark W. Spong, Seth Hutchinson and M. Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, 2006. ISBN: 0-471-64990-2.

[46] Stratasys. *3D Printing With ABSplus*. URL: http://www.stratasys.com/materials/fdm/absplus.

[47] Stratasys. *Fortus 250mc*. URL: http://www.stratasys.com/3d-printers/design-series/fortus-250mc.

[48] Sheffield Telegraph. *Solid finds solution in business park switch*. URL: http://www.sheffieldtelegraph.co.uk/news/business/solid-finds-solution-in-business-park-switch-1-458019.

[49] TrossenRobotics. *Projects Using the AX-12 Dynamixel Servos & Brackets*. URL: http://www.trossenrobotics.com/dynamixel-ax-12-robot-actuator.aspx.

[50] TrossenRobotics. *Strength & Speed Chart of the Dynamixel Family*. URL: http://www.trossenrobotics.com/dynamixel-ax-12-robot-actuator.aspx.

[51] Ozgur Unver and M. Sitti. 'Flat Dry Elastomer Adhesives as Attachment Materials for Climbing Robots'. In: *Robotics, IEEE Transactions on* 26.1 (Feb. 2010), pp. 131–141. ISSN: 1552-3098. DOI: 10.1109/TRO.2009.2033628.

[52] Shanqiang Wu et al. 'A Wireless Distributed Wall Climbing Robotic System for Reconnaissance Purpose'. In: *Mechatronics and Automation, Proceedings of the 2006 IEEE International Conference on*. June 2006, pp. 1308–1312. DOI: 10.1109/ICMA.2006.257816.

[53] Wang Yan et al. 'Development and application of wall-climbing robots'. In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 2. 1999, 1207–1212 vol.2. DOI: 10.1109/ROBOT.1999.772526.

Appendix A

KlimBot Control System source code

```
package KlimBotPack;

import SimpleDynamixel.*;
import java.awt.ComponentOrientation;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import java.util.ArrayList;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;
import javax.swing.SwingWorker;
import processing.core.PApplet;
import processing.serial.*;

/**
 *
 * @author Kim
 */
public class KlimBotControl extends javax.swing.JFrame {

    /**
     * Creates new form KlimBotControl
     */
   public KlimBotControl() {
      initComponents();

        initDemoWall();
        //initTestWall();

        //Scaling images to match wall size
        Image img = imgBolt.getImage() ;
Image newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgBolt = new ImageIcon(newimg);
img = imgUnreachableBolt.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgUnreachableBolt = new ImageIcon(newimg);
img = imgChosenBolt.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgChosenBolt = new ImageIcon(newimg);
img = imgKlimBotBase.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotBase = new ImageIcon(newimg);
img = imgKlimBotLegUp.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotLegUp = new ImageIcon(newimg);
img = imgKlimBotLegLeft.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotLegLeft = new ImageIcon(newimg);
img = imgKlimBotLegRight.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotLegRight = new ImageIcon(newimg);
img = imgKlimBotLegDown.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotLegDown = new ImageIcon(newimg);
img = imgKlimBotThighUp.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotThighUp = new ImageIcon(newimg);
img = imgKlimBotThighLeft.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotThighLeft = new ImageIcon(newimg);
img = imgKlimBotThighRight.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotThighRight = new ImageIcon(newimg);
img = imgKlimBotThighDown.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotThighDown = new ImageIcon(newimg);
img = imgKlimBotOutstretchedInnerLegUp.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotOutstretchedInnerLegUp = new ImageIcon(newimg);
img = imgKlimBotOutstretchedInnerLegLeft.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
```

```
imgKlimBotOutstretchedInnerLegLeft = new ImageIcon(newimg);
img = imgKlimBotOutstretchedInnerLegRight.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotOutstretchedInnerLegRight = new ImageIcon(newimg);
img = imgKlimBotOutstretchedInnerLegDown.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotOutstretchedInnerLegDown = new ImageIcon(newimg);
img = imgKlimBotOutstretchedOuterLegUp.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotOutstretchedOuterLegUp = new ImageIcon(newimg);
img = imgKlimBotOutstretchedOuterLegLeft.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotOutstretchedOuterLegLeft = new ImageIcon(newimg);
img = imgKlimBotOutstretchedOuterLegRight.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotOutstretchedOuterLegRight = new ImageIcon(newimg);
img = imgKlimBotOutstretchedOuterLegDown.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotOutstretchedOuterLegDown = new ImageIcon(newimg);
img = imgKlimBotOutstretchedThighUp.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotOutstretchedThighUp = new ImageIcon(newimg);
img = imgKlimBotOutstretchedThighLeft.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotOutstretchedThighLeft = new ImageIcon(newimg);
img = imgKlimBotOutstretchedThighRight.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotOutstretchedThighRight = new ImageIcon(newimg);
img = imgKlimBotOutstretchedThighDown.getImage() ;
newimg = img.getScaledInstance(imageSize, imageSize,
    java.awt.Image.SCALE_SMOOTH);
imgKlimBotOutstretchedThighDown = new ImageIcon(newimg);


comNmb = "COM" + JOptionPane.showInputDialog(null, "Enter
    COM-port number");

//initiate Processing sketch
javax.swing.JPanel panel = new javax.swing.JPanel();
panel.setBounds(0, 0, 0, 0);
sketch = new ProcessingSketch();
panel.add(sketch);
this.add(panel);
sketch.init(); //this is the function used to start the
    execution of the sketch
```

```
this.setVisible(true);


//draw bolts
for (int i = 0; i < squareGrid.length; i++) {
 for (int j = 0; j < squareGrid[0].length; j++) {
       if (squareGrid[i][j].hasBolt) {
           //draws reachable bolts black, unreachable bolts
               gray
           if (i+4 < squareGrid.length && j+2 <
               squareGrid[0].length && j >= 2 &&
                   squareGrid[i+2][j-2].hasBolt &&
                       squareGrid[i+2][j+2].hasBolt &&
                       squareGrid[i+4][j].hasBolt) {
               squareGrid[i][j].background = imgBolt;
               squareGrid[i][j].setIcon(squareGrid[i][j].background);
               squareGrid[i][j].isReachable = true;
               reachableSquaresCnt++;
               // for Breadth-First-Search, storing
                   neighbouring nodes
               if (i != 0 && squareGrid[i-1][j].isReachable)
                   {
                   squareGrid[i][j].children[0] =
                       squareGrid[i-1][j]; //up
                   squareGrid[i-1][j].children[1] =
                       squareGrid[i][j]; //down
               }
               if (i != squareGrid[0].length-1 &&
                   squareGrid[i][j-1].isReachable) {
                   squareGrid[i][j].children[3] =
                       squareGrid[i][j-1]; //left
                   squareGrid[i][j-1].children[2] =
                       squareGrid[i][j]; //right
               }
           } else {
               squareGrid[i][j].background =
                   imgUnreachableBolt;
               squareGrid[i][j].setIcon(squareGrid[i][j].background);
           }
       }
       squareGridPanel.add(squareGrid[i][j]);
   }
}

//Place KlimBot on the wall
presentPositionRow = 3;
presentPositionColumn = 4;
drawKlimBot();

//initialize buttons
if (isReachable(presentPositionRow-1,
    presentPositionColumn)) {
    upButton.setEnabled(true);
} else {
```

```
            upButton.setEnabled(false);
        }
        if (isReachable(presentPositionRow,
            presentPositionColumn-1)) {
            leftButton.setEnabled(true);
        } else {
            leftButton.setEnabled(false);
        }
        if (isReachable(presentPositionRow,
            presentPositionColumn+1)) {
            rightButton.setEnabled(true);
        } else {
            rightButton.setEnabled(false);
        }
        if (isReachable(presentPositionRow+1,
            presentPositionColumn)) {
            downButton.setEnabled(true);
        } else {
            downButton.setEnabled(false);
        }
    }


    void initTestWall() {
      imageSize = 40;
        squareGrid = new Square[10][7];
        squareGridScrollPane.setMaximumSize(new
            java.awt.Dimension(280, 400));
        squareGridScrollPane.setMinimumSize(new
            java.awt.Dimension(280, 400)); //280, 400
        squareGridScrollPane.setPreferredSize(new
            java.awt.Dimension(280, 400));
        squareGridPanel.setMaximumSize(new java.awt.Dimension(280,
            400)); //800 800
        squareGridPanel.setPreferredSize(new
            java.awt.Dimension(7*imageSize, 10*imageSize));
        squareGridPanel.setLayout(new java.awt.GridLayout(10, 7));
        squareGridScrollPane.setViewportView(squareGridPanel);
        squareGridPanel.setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);

       for (int i = 0; i < squareGrid.length; i++) {
            for (int j = 0; j < squareGrid[0].length; j++) {
                squareGrid[i][j] = new Square();
            }
        }

        //for bolt configuration
        squareGrid[0][0].hasBolt = false;
        squareGrid[0][1].hasBolt = false;
        squareGrid[1][0].hasBolt = false;
        squareGrid[1][1].hasBolt = false;
        squareGrid[0][5].hasBolt = false;
        squareGrid[0][6].hasBolt = false;
        squareGrid[1][5].hasBolt = false;
        squareGrid[1][6].hasBolt = false;
        squareGrid[8][0].hasBolt = false;
        squareGrid[8][1].hasBolt = false;
        squareGrid[9][0].hasBolt = false;
        squareGrid[9][1].hasBolt = false;
        squareGrid[8][5].hasBolt = false;
        squareGrid[8][6].hasBolt = false;
        squareGrid[9][5].hasBolt = false;
        squareGrid[9][6].hasBolt = false;
    }

    void initDemoWall() {
        imageSize = 20;
        squareGrid = new Square[20][20];
        squareGridScrollPane.setMaximumSize(new
            java.awt.Dimension(280, 400));
        squareGridScrollPane.setMinimumSize(new
            java.awt.Dimension(280, 400)); //280, 400
        squareGridScrollPane.setPreferredSize(new
            java.awt.Dimension(280, 400));
        squareGridPanel.setMaximumSize(new java.awt.Dimension(600,
            600)); //800 800
        squareGridPanel.setPreferredSize(new
            java.awt.Dimension(20*imageSize, 20*imageSize-9));
        squareGridPanel.setLayout(new
            java.awt.GridLayout(squareGrid[0].length,
            squareGrid.length));
        squareGridScrollPane.setViewportView(squareGridPanel);

        //fills the grid with square without bolts
        squareGridPanel.setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);
        for (int i = 0; i < squareGrid.length; i++) {
            for (int j = 0; j < squareGrid[0].length; j++) {
                squareGrid[i][j] = new Square();
                squareGrid[i][j].hasBolt = false;
            }
        }
        //place bolts
        for (int i = 2; i < 18; i++) {
            squareGrid[i][0].hasBolt = true;
        }
        for (int i = 2; i < 7; i++) {
            squareGrid[i][1].hasBolt = true;
        }
        for (int i = 0; i < 20; i++) {
            squareGrid[i][2].hasBolt = true;
        }
        for (int i = 0; i < 9; i++) {
            squareGrid[i][3].hasBolt = true;
        }
        for (int i = 0; i < 18; i++) {
            squareGrid[i][4].hasBolt = true;
        }
        for (int i = 2; i < 7; i++) {
            squareGrid[i][5].hasBolt = true;
            squareGrid[i][6].hasBolt = true;
            squareGrid[i][7].hasBolt = true;
            squareGrid[i][8].hasBolt = true;
            squareGrid[i][9].hasBolt = true;
            squareGrid[i][10].hasBolt = true;
            squareGrid[i][11].hasBolt = true;
            squareGrid[i][12].hasBolt = true;
            squareGrid[i][13].hasBolt = true;
            squareGrid[i][14].hasBolt = true;
        }
        squareGrid[3][7].hasBolt = false;
        squareGrid[3][8].hasBolt = false;
        squareGrid[3][9].hasBolt = false;
        squareGrid[5][7].hasBolt = false;
        squareGrid[2][13].hasBolt = false;
        squareGrid[2][14].hasBolt = false;
        squareGrid[3][13].hasBolt = false;
        squareGrid[3][14].hasBolt = false;
        for (int j = 10; j < 18; j++) {
            squareGrid[7][j].hasBolt = true;
        }
        for (int j = 10; j < 17; j++) {
            squareGrid[8][j].hasBolt = true;
        }
        squareGrid[8][13].hasBolt = false;
        for (int j = 10; j < 20; j++) {
            squareGrid[9][j].hasBolt = true;
        }
        for (int j = 12; j < 15; j++) {
            squareGrid[10][j].hasBolt = true;
        }
        for (int j = 12; j < 18; j++) {
            squareGrid[11][j].hasBolt = true;
        }
    }

   //draws a graphical representation of KlimBot
    void drawKlimBot() {
        squareGrid[presentPositionRow][presentPositionColumn]
         .drawImage(imgKlimBotLegUp, false);
        squareGrid[presentPositionRow+1][presentPositionColumn]
         .drawImage(imgKlimBotThighUp, true);
        squareGrid[presentPositionRow+2][presentPositionColumn]
         .drawImage(imgKlimBotBase, true);
        squareGrid[presentPositionRow+3][presentPositionColumn]
         .drawImage(imgKlimBotThighDown, true);
        squareGrid[presentPositionRow+4][presentPositionColumn]
         .drawImage(imgKlimBotLegDown, false);
        squareGrid[presentPositionRow+2][presentPositionColumn-2]
         .drawImage(imgKlimBotLegLeft, false);
        squareGrid[presentPositionRow+2][presentPositionColumn-1]
         .drawImage(imgKlimBotThighLeft, true);
        squareGrid[presentPositionRow+2][presentPositionColumn+1]
        .drawImage(imgKlimBotThighRight, true);
    squareGrid[presentPositionRow+2][presentPositionColumn+2]
        .drawImage(imgKlimBotLegRight, false);
}

void drawKlimBotReachUp() {
    squareGrid[presentPositionRow-1][presentPositionColumn]
        .drawImage(imgKlimBotOutstretchedOuterLegUp, false);
    squareGrid[presentPositionRow][presentPositionColumn]
        .drawImage(imgKlimBotOutstretchedInnerLegUp, false);
    squareGrid[presentPositionRow+1][presentPositionColumn]
        .drawImage(imgKlimBotOutstretchedThighUp, true);
    presentPositionRow--;
}

void drawKlimBotDragUp() {
    //remove previous drawing
    squareGrid[presentPositionRow+3][presentPositionColumn-2]
        .removeForground();
    squareGrid[presentPositionRow+3][presentPositionColumn-1]
        .removeForground();
    squareGrid[presentPositionRow+3][presentPositionColumn+1]
        .removeForground();
    squareGrid[presentPositionRow+3][presentPositionColumn+2]
        .removeForground();
    //draw
    squareGrid[presentPositionRow][presentPositionColumn]
        .drawImage(imgKlimBotLegUp, false);
    squareGrid[presentPositionRow+1][presentPositionColumn]
        .drawImage(imgKlimBotThighUp, true);
    squareGrid[presentPositionRow+2][presentPositionColumn]
        .drawImage(imgKlimBotBase, true);
    squareGrid[presentPositionRow+2][presentPositionColumn-2]
        .drawImage(imgKlimBotLegLeft, false);
    squareGrid[presentPositionRow+2][presentPositionColumn-1]
        .drawImage(imgKlimBotThighLeft, true);
    squareGrid[presentPositionRow+2][presentPositionColumn+1]
        .drawImage(imgKlimBotThighRight, true);
    squareGrid[presentPositionRow+2][presentPositionColumn+2]
        .drawImage(imgKlimBotLegRight, false);
    squareGrid[presentPositionRow+3][presentPositionColumn]
        .drawImage(imgKlimBotOutstretchedThighDown, true);
    squareGrid[presentPositionRow+4][presentPositionColumn]
        .drawImage(imgKlimBotOutstretchedInnerLegDown, false);
    squareGrid[presentPositionRow+5][presentPositionColumn]
        .drawImage(imgKlimBotOutstretchedOuterLegDown, false);
}

//assumes KlimBot is already drawn underneath
void drawKlimBotRetractUp() {
    //remove previous drawing
    squareGrid[presentPositionRow+5][presentPositionColumn]
        .removeForground();
    //draw
```

```
        squareGrid[presentPositionRow+3][presentPositionColumn]
         .drawImage(imgKlimBotThighDown, true);
        squareGrid[presentPositionRow+4][presentPositionColumn]
         .drawImage(imgKlimBotLegDown, false);
    }

    //assumes KlimBot is already drawn above
    void drawKlimBotReachDown() {
        squareGrid[presentPositionRow+3][presentPositionColumn]
         .drawImage(imgKlimBotOutstretchedThighDown, true);
        squareGrid[presentPositionRow+4][presentPositionColumn]
         .drawImage(imgKlimBotOutstretchedInnerLegDown, false);
        squareGrid[presentPositionRow+5][presentPositionColumn]
         .drawImage(imgKlimBotOutstretchedOuterLegDown, false);
    }

    void drawKlimBotDragDown() {
        //remove previous drawing
        squareGrid[presentPositionRow+2][presentPositionColumn-2]
         .removeForground();
        squareGrid[presentPositionRow+2][presentPositionColumn-1]
         .removeForground();
        squareGrid[presentPositionRow+2][presentPositionColumn+1]
         .removeForground();
        squareGrid[presentPositionRow+2][presentPositionColumn+2]
         .removeForground();
        //draw
        squareGrid[presentPositionRow+5][presentPositionColumn]
         .drawImage(imgKlimBotLegDown, false);
        squareGrid[presentPositionRow+4][presentPositionColumn]
         .drawImage(imgKlimBotThighDown, true);
        squareGrid[presentPositionRow+3][presentPositionColumn]
         .drawImage(imgKlimBotBase, true);
        squareGrid[presentPositionRow+3][presentPositionColumn-2]
         .drawImage(imgKlimBotLegLeft, false);
        squareGrid[presentPositionRow+3][presentPositionColumn-1]
         .drawImage(imgKlimBotThighLeft, true);
        squareGrid[presentPositionRow+3][presentPositionColumn+1]
         .drawImage(imgKlimBotThighRight, true);
        squareGrid[presentPositionRow+3][presentPositionColumn+2]
         .drawImage(imgKlimBotLegRight, false);
        squareGrid[presentPositionRow+2][presentPositionColumn]
         .drawImage(imgKlimBotOutstretchedThighUp, true);
        squareGrid[presentPositionRow+1][presentPositionColumn]
         .drawImage(imgKlimBotOutstretchedInnerLegUp, false);
        squareGrid[presentPositionRow][presentPositionColumn]
         .drawImage(imgKlimBotOutstretchedOuterLegUp, false);
    }

    void drawKlimBotRetractDown() {
        //remove previous drawing
        squareGrid[presentPositionRow][presentPositionColumn]
         .removeForground();
        //draw
    squareGrid[presentPositionRow+2][presentPositionColumn]
     .drawImage(imgKlimBotThighUp, true);
    squareGrid[presentPositionRow+1][presentPositionColumn]
     .drawImage(imgKlimBotLegUp, false);
    presentPositionRow++;
}

//assumes KlimBot is already drawn to the left
void drawKlimBotReachRight() {
    squareGrid[presentPositionRow+2][presentPositionColumn+1]
     .drawImage(imgKlimBotOutstretchedThighRight, true);
    squareGrid[presentPositionRow+2][presentPositionColumn+2]
     .drawImage(imgKlimBotOutstretchedInnerLegRight, false);
    squareGrid[presentPositionRow+2][presentPositionColumn+3]
     .drawImage(imgKlimBotOutstretchedOuterLegRight, false);
}

void drawKlimBotDragRight() {
    //remove previuos drawing
    squareGrid[presentPositionRow][presentPositionColumn]
     .removeForground();
    squareGrid[presentPositionRow+1][presentPositionColumn]
     .removeForground();
    squareGrid[presentPositionRow+3][presentPositionColumn]
     .removeForground();
    squareGrid[presentPositionRow+4][presentPositionColumn]
     .removeForground();
    //draw
    squareGrid[presentPositionRow+2][presentPositionColumn+3]
     .drawImage(imgKlimBotLegRight, false);
    squareGrid[presentPositionRow+2][presentPositionColumn+2]
     .drawImage(imgKlimBotThighRight, true);
    squareGrid[presentPositionRow+2][presentPositionColumn+1]
     .drawImage(imgKlimBotBase, true);
    squareGrid[presentPositionRow][presentPositionColumn+1]
     .drawImage(imgKlimBotLegUp, false);
    squareGrid[presentPositionRow+1][presentPositionColumn+1]
     .drawImage(imgKlimBotThighUp, true);
    squareGrid[presentPositionRow+3][presentPositionColumn+1]
     .drawImage(imgKlimBotThighDown, true);
    squareGrid[presentPositionRow+4][presentPositionColumn+1]
     .drawImage(imgKlimBotLegDown, false);
    squareGrid[presentPositionRow+2][presentPositionColumn]
     .drawImage(imgKlimBotOutstretchedThighLeft, true);
    squareGrid[presentPositionRow+2][presentPositionColumn-1]
     .drawImage(imgKlimBotOutstretchedInnerLegLeft, false);
    squareGrid[presentPositionRow+2][presentPositionColumn-2]
     .drawImage(imgKlimBotOutstretchedOuterLegLeft, false);
    presentPositionColumn++;
}

void drawKlimBotRetractRight() {
    //remove previous drawing
    squareGrid[presentPositionRow+2][presentPositionColumn-3]
```

```
        .removeForground();
    //draw
    squareGrid[presentPositionRow+2][presentPositionColumn-1]
        .drawImage(imgKlimBotThighLeft, true);
    squareGrid[presentPositionRow+2][presentPositionColumn-2]
        .drawImage(imgKlimBotLegLeft, false);
}

//assumes KlimBot is already drawn to the right
void drawKlimBotReachLeft() {
    squareGrid[presentPositionRow+2][presentPositionColumn-1]
        .drawImage(imgKlimBotOutstretchedThighLeft, true);
    squareGrid[presentPositionRow+2][presentPositionColumn-2]
        .drawImage(imgKlimBotOutstretchedInnerLegLeft, false);
    squareGrid[presentPositionRow+2][presentPositionColumn-3]
        .drawImage(imgKlimBotOutstretchedOuterLegLeft, false);
}

void drawKlimBotDragLeft() {
    //removes previous drawing
    squareGrid[presentPositionRow][presentPositionColumn]
        .removeForground();
    squareGrid[presentPositionRow+1][presentPositionColumn]
        .removeForground();
    squareGrid[presentPositionRow+3][presentPositionColumn]
        .removeForground();
    squareGrid[presentPositionRow+4][presentPositionColumn]
        .removeForground();
    //draw
    squareGrid[presentPositionRow+2][presentPositionColumn-3]
        .drawImage(imgKlimBotLegLeft, false);
    squareGrid[presentPositionRow+2][presentPositionColumn-2]
        .drawImage(imgKlimBotThighLeft, true);
    squareGrid[presentPositionRow+2][presentPositionColumn-1]
        .drawImage(imgKlimBotBase, true);
    squareGrid[presentPositionRow][presentPositionColumn-1]
        .drawImage(imgKlimBotLegUp, false);
    squareGrid[presentPositionRow+1][presentPositionColumn-1]
        .drawImage(imgKlimBotThighUp, true);
    squareGrid[presentPositionRow+3][presentPositionColumn-1]
        .drawImage(imgKlimBotThighDown, true);
    squareGrid[presentPositionRow+4][presentPositionColumn-1]
        .drawImage(imgKlimBotLegDown, false);
    squareGrid[presentPositionRow+2][presentPositionColumn]
        .drawImage(imgKlimBotOutstretchedThighRight, true);
    squareGrid[presentPositionRow+2][presentPositionColumn+1]
        .drawImage(imgKlimBotOutstretchedInnerLegRight, false);
    squareGrid[presentPositionRow+2][presentPositionColumn+2]
        .drawImage(imgKlimBotOutstretchedOuterLegRight, false);
    presentPositionColumn--;
}

void drawKlimBotRetractLeft() {
    //remove previous drawing
```

```
        squareGrid[presentPositionRow+2][presentPositionColumn+3]
         .removeForground();
        //draw
        squareGrid[presentPositionRow+2][presentPositionColumn+1]
         .drawImage(imgKlimBotThighRight, true);
        squareGrid[presentPositionRow+2][presentPositionColumn+2]
         .drawImage(imgKlimBotLegRight, false);
    }


    @SuppressWarnings("unchecked")
    private void initComponents() {
        pathChoiceGroup = new javax.swing.ButtonGroup();
        rightButton = new javax.swing.JButton();
        downButton = new javax.swing.JButton();
        leftButton = new javax.swing.JButton();
        upButton = new javax.swing.JButton();
        abortButton = new javax.swing.JButton();
        manualControlButton = new javax.swing.JRadioButton();
        allBoltsButton = new javax.swing.JRadioButton();
        singleBoltButton = new javax.swing.JRadioButton();
        choosePathButton = new javax.swing.JRadioButton();
        stopButton = new javax.swing.JButton();
        startButton = new javax.swing.JButton();
        squareGridScrollPane = new javax.swing.JScrollPane();
        squareGridPanel = new javax.swing.JPanel();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("KlimBot Control System");
        setBackground(new java.awt.Color(255, 255, 255));
        setResizable(false);

        rightButton.setIcon(new javax.swing.ImageIcon(getClass()
         .getResource("/KlimBotPack/rightArrow.png"))); // NOI18N
        rightButton.setFocusable(false);
        rightButton.setMaximumSize(new java.awt.Dimension(50, 50));
        rightButton.setMinimumSize(new java.awt.Dimension(50, 50));
        rightButton.setPreferredSize(new java.awt.Dimension(50, 50));
        rightButton.addActionListener(new
            java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
                evt) {
                rightButtonActionPerformed(evt);
            }
        });

        downButton.setIcon(new javax.swing.ImageIcon(getClass()
         .getResource("/KlimBotPack/downArrow.png"))); // NOI18N
        downButton.setFocusable(false);
        downButton.setMaximumSize(new java.awt.Dimension(50, 50));
        downButton.setMinimumSize(new java.awt.Dimension(50, 50));
        downButton.setPreferredSize(new java.awt.Dimension(50, 50));
        downButton.addActionListener(new
            java.awt.event.ActionListener() {
```

```
    public void actionPerformed(java.awt.event.ActionEvent
        evt) {
        downButtonActionPerformed(evt);
    }
});

leftButton.setIcon(new javax.swing.ImageIcon(getClass()
 .getResource("/KlimBotPack/leftArrow.png"))); // NOI18N
leftButton.setFocusable(false);
leftButton.setMaximumSize(new java.awt.Dimension(50, 50));
leftButton.setMinimumSize(new java.awt.Dimension(50, 50));
leftButton.setPreferredSize(new java.awt.Dimension(50, 50));
leftButton.addActionListener(new
    java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent
        evt) {
        leftButtonActionPerformed(evt);
    }
});

upButton.setIcon(new javax.swing.ImageIcon(getClass()
 .getResource("/KlimBotPack/upArrow.png"))); // NOI18N
upButton.setFocusable(false);
upButton.setMaximumSize(new java.awt.Dimension(50, 50));
upButton.setMinimumSize(new java.awt.Dimension(50, 50));
upButton.setPreferredSize(new java.awt.Dimension(50, 50));
upButton.addActionListener(new
    java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent
        evt) {
        upButtonActionPerformed(evt);
    }
});

abortButton.setIcon(new javax.swing.ImageIcon(getClass()
 .getResource("/KlimBotPack/abort.png"))); // NOI18N
abortButton.setEnabled(false);
abortButton.setFocusable(false);
abortButton.setMaximumSize(new java.awt.Dimension(111, 111));
abortButton.setMinimumSize(new java.awt.Dimension(111, 111));
abortButton.setPreferredSize(new java.awt.Dimension(111,
    111));
abortButton.addActionListener(new
    java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent
        evt) {
        abortButtonActionPerformed(evt);
    }
});

pathChoiceGroup.add(manualControlButton);
manualControlButton.setSelected(true);
manualControlButton.setText("Manual control");
manualControlButton.setFocusable(false);
manualControlButton.addActionListener(new
    java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent
        evt) {
        manualControlButtonActionPerformed(evt);
    }
});

pathChoiceGroup.add(allBoltsButton);
allBoltsButton.setText("Traverse all bolts");
allBoltsButton.setFocusable(false);
allBoltsButton.addActionListener(new
    java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent
        evt) {
        allBoltsButtonActionPerformed(evt);
    }
});

pathChoiceGroup.add(singleBoltButton);
singleBoltButton.setText("Visit single bolt");
singleBoltButton.setFocusable(false);
singleBoltButton.addActionListener(new
    java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent
        evt) {
        singleBoltButtonActionPerformed(evt);
    }
});

pathChoiceGroup.add(choosePathButton);
choosePathButton.setText("Specify path");
choosePathButton.setFocusable(false);
choosePathButton.addActionListener(new
    java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent
        evt) {
        choosePathButtonActionPerformed(evt);
    }
});

stopButton.setIcon(new javax.swing.ImageIcon(getClass()
 .getResource("/KlimBotPack/stop.png"))); // NOI18N
stopButton.setEnabled(false);
stopButton.setFocusable(false);
stopButton.setMaximumSize(new java.awt.Dimension(50, 50));
stopButton.setMinimumSize(new java.awt.Dimension(50, 50));
stopButton.setPreferredSize(new java.awt.Dimension(50, 50));
stopButton.addActionListener(new
    java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent
        evt) {
        stopButtonActionPerformed(evt);
    }
```

```
        });

        startButton.setIcon(new javax.swing.ImageIcon(getClass()
         .getResource("/KlimBotPack/start.png"))); // NOI18N
        startButton.setEnabled(false);
        startButton.setFocusable(false);
        startButton.setMaximumSize(new java.awt.Dimension(50, 50));
        startButton.setMinimumSize(new java.awt.Dimension(50, 50));
        startButton.setPreferredSize(new java.awt.Dimension(50, 50));
        startButton.addActionListener(new
            java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
                evt) {
                startButtonActionPerformed(evt);
            }
        });

        squareGridScrollPane.setMaximumSize(new
            java.awt.Dimension(3000, 4000));
        squareGridScrollPane.setMinimumSize(new
            java.awt.Dimension(280, 400));
        squareGridScrollPane.setPreferredSize(new
            java.awt.Dimension(280, 400));

        squareGridPanel.setMaximumSize(new java.awt.Dimension(800,
            800));
        squareGridPanel.setPreferredSize(new java.awt.Dimension(280,
            400));
        squareGridPanel.setLayout(new java.awt.GridLayout(10, 7));
        squareGridScrollPane.setViewportView(squareGridPanel);

javax.swing.GroupLayout layout = new
    javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
        layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(squareGridScrollPane,
            javax.swing.GroupLayout.PREFERRED_SIZE, 290,
            javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle
         .ComponentPlacement.RELATED, 44, Short.MAX_VALUE)
        .addGroup(layout.createParallelGroup(javax.swing
         .GroupLayout.Alignment.LEADING, false)
            .addGroup(layout.createSequentialGroup()
                .addComponent(abortButton,
                    javax.swing.GroupLayout.PREFERRED_SIZE,
                    javax.swing.GroupLayout.DEFAULT_SIZE,
                    javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addComponent(leftButton,
                    javax.swing.GroupLayout.PREFERRED_SIZE,
                    javax.swing.GroupLayout.DEFAULT_SIZE,
                    javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing
               .GroupLayout.Alignment.LEADING)
                    .addComponent(downButton,
                        javax.swing.GroupLayout.PREFERRED_SIZE,
                        javax.swing.GroupLayout.DEFAULT_SIZE,
                        javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(upButton,
                        javax.swing.GroupLayout.PREFERRED_SIZE,
                        javax.swing.GroupLayout.DEFAULT_SIZE,
                        javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(rightButton,
                    javax.swing.GroupLayout.PREFERRED_SIZE,
                    javax.swing.GroupLayout.DEFAULT_SIZE,
                    javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing
               .GroupLayout.Alignment.LEADING, false)
                    .addComponent(manualControlButton)
                    .addComponent(allBoltsButton)
                    .addComponent(singleBoltButton)
                    .addComponent(choosePathButton))
                .addPreferredGap(javax.swing.LayoutStyle
               .ComponentPlacement.RELATED,
               javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(startButton,
                    javax.swing.GroupLayout.PREFERRED_SIZE,
                    javax.swing.GroupLayout.DEFAULT_SIZE,
                    javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addComponent(stopButton,
                    javax.swing.GroupLayout.PREFERRED_SIZE,
                    javax.swing.GroupLayout.DEFAULT_SIZE,
                    javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(30, 30, 30))))
);
layout.setVerticalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(layout.createParallelGroup(javax.swing
         .GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(0, 141, Short.MAX_VALUE)
                .addComponent(manualControlButton)
                .addPreferredGap(javax.swing.LayoutStyle
               .ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing
               .GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                    .addComponent(allBoltsButton)
                    .addPreferredGap(javax.swing.LayoutStyle
                 .ComponentPlacement.RELATED)
                    .addComponent(singleBoltButton)
                    .addPreferredGap(javax.swing.LayoutStyle
                 .ComponentPlacement.RELATED)
                    .addComponent(choosePathButton)
                    .addGroup(layout.createParallelGroup(javax.swing
                 .GroupLayout.Alignment.LEADING)
                        .addGroup(layout.createSequentialGroup()
                            .addGap(38, 38, 38)
                            .addGroup(layout.createParallelGroup(javax
                       .swing.GroupLayout.Alignment.LEADING)
                                .addGroup(layout.createSequentialGroup()
                                    .addComponent(upButton,
                                        javax.swing
                             .GroupLayout.PREFERRED_SIZE,
                             javax.swing.GroupLayout.DEFAULT_SIZE,
                             javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addPreferredGap(javax.swing.LayoutStyle
                             .ComponentPlacement.UNRELATED)
                                    .addComponent(downButton,
                             javax.swing.GroupLayout.PREFERRED_SIZE,
                             javax.swing.GroupLayout.DEFAULT_SIZE,
                             javax.swing.GroupLayout.PREFERRED_SIZE))
                                .addComponent(abortButton,
                          javax.swing.GroupLayout.PREFERRED_SIZE,
                          javax.swing.GroupLayout.DEFAULT_SIZE,
                          javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addGap(31, 31, 31))
                        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                    layout.createSequentialGroup()
                            .addPreferredGap(javax.swing.LayoutStyle
                       .ComponentPlacement.RELATED)
                            .addGroup(layout.createParallelGroup(javax
                       .swing.GroupLayout.Alignment.LEADING)
                                .addComponent(leftButton,
                          javax.swing.GroupLayout.Alignment.TRAILING,
                          javax.swing.GroupLayout.PREFERRED_SIZE,
                          javax.swing.GroupLayout.DEFAULT_SIZE,
                          javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(rightButton,
                          javax.swing.GroupLayout.Alignment.TRAILING,
                          javax.swing.GroupLayout.PREFERRED_SIZE,
                          javax.swing.GroupLayout.DEFAULT_SIZE,
                          javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addGap(62, 62, 62))))
                .addGroup(layout.createParallelGroup(javax.swing
              .GroupLayout.Alignment.TRAILING)
                    .addComponent(startButton,
                 javax.swing.GroupLayout.PREFERRED_SIZE,
                 javax.swing.GroupLayout.DEFAULT_SIZE,
                 javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(stopButton,
                    javax.swing.GroupLayout.PREFERRED_SIZE,
                    javax.swing.GroupLayout.DEFAULT_SIZE,
                    javax.swing.GroupLayout.PREFERRED_SIZE))))
            .addComponent(squareGridScrollPane,
            javax.swing.GroupLayout.DEFAULT_SIZE,
            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        .addContainerGap())
);

        pack();
        setLocationRelativeTo(null);
    }

    private void
        allBoltsButtonActionPerformed(java.awt.event.ActionEvent
        evt) {
        enableAllButtons(false);
        cleanBolts();
        radioButton = RadioButtons.ALL;
        visitedBoltsCnt = 0;
        startButton.setEnabled(true);
    }

    private void upButtonActionPerformed(java.awt.event.ActionEvent
        evt) {
        if (isReachable(presentPositionRow-1,
            presentPositionColumn)) {
            direction = Directions.UP;
            (directionWorker = new MoveWorker()).execute();
            directionButtonPressedEnabling(true);
        } else {
            System.out.println("KlimBot can't move up");
        }
    }

    //Thread for performing a move
    private class MoveWorker extends SwingWorker<Void, Void> {

    public MoveWorker() {
    }

    @Override
    protected Void doInBackground() {

        isMoving = true;
        abortButton.setEnabled(true);
        switch (direction) {
            case UP:
                sketch.moveUp();
                break;
            case DOWN:
                sketch.moveDown();
                break;
            case LEFT:
```

```
            sketch.moveLeft();
            break;
        case RIGHT:
            sketch.moveRight();
            break;
    }
    return null;
}

@Override
protected void done() {
    isMoving = false;
    switch (direction) {
        case UP:
            System.out.println("KlimBot has moved up");
            break;
        case DOWN:
            System.out.println("KlimBot has moved down");
            break;
        case LEFT:
            System.out.println("KlimBot has moved left");
            break;
        case RIGHT:
            System.out.println("KlimBot has moved right");
            break;
    }


    if (radioButton == RadioButtons.MANUAL){
        directionButtonPressedEnabling(false);
    } else /* if (radioButton == RadioButtons.SINGLE) */ {
        // movePath();
        //visitedBoltsCnt++;
        if (!stopPressed) {
            if (!pathDirections.isEmpty()) {
                direction =
                    pathDirections.remove(pathDirections.size()-1);
                (directionWorker = new MoveWorker()).execute();
                if (pathDirections.isEmpty()) {
                    if (!(radioButton == RadioButtons.PATH &&
                 !pathSquares.isEmpty())) {
                        stopButton.setEnabled(false);
                    }
                }
            } else {
                visitedBoltsCnt++;
                if (radioButton == RadioButtons.ALL) {
                 //unless all bolts are visited, search again
                    if (visitedBoltsCnt < reachableSquaresCnt) {
                        (pathPlannerWorker =
                    new PathPlannerWorker()).execute();
                    } else { // all bolts visited
                        stopButton.setEnabled(false);
                        abortButton.setEnabled(false);
                        startButton.setEnabled(true);
                        visitedBoltsCnt = 0;
                    }
                } else if (radioButton == RadioButtons.PATH) {
                    squareGrid[presentPositionRow][presentPositionColumn]
                 .background = imgBolt;
                    squareGrid[presentPositionRow][presentPositionColumn]
                 .setIcon(combinedIcons(squareGrid
                  [presentPositionRow][presentPositionColumn].background,
                        squareGrid[presentPositionRow]
                 [presentPositionColumn].forground));
                    if (!pathSquares.isEmpty()) {
                        (pathPlannerWorker = new
                            PathPlannerWorker()).execute();
                    } else {
                        stopButton.setEnabled(false);
                        abortButton.setEnabled(false);
                    }
                } else {

                System.out.println("all move-threads are done");

                cleanBolts();
                abortButton.setEnabled(false);

                }

            }
        } else { //for restarting
            stopPressed = false;
            startButton.setEnabled(true);
            abortButton.setEnabled(false);
            if (radioButton == RadioButtons.PATH &&
           pathDirections.isEmpty()) {
                squareGrid[presentPositionRow][presentPositionColumn]
              .background = imgBolt;
                squareGrid[presentPositionRow][presentPositionColumn]
              .setIcon(combinedIcons(squareGrid[presentPositionRow]
              [presentPositionColumn].background,
                    squareGrid[presentPositionRow][presentPositionColumn]
              .forground));
            }
        }
    }
    }
}

//Thread for generating path
private class PathPlannerWorker extends SwingWorker<Void, Void> {

    public PathPlannerWorker() {
    }

    @Override
    protected Void doInBackground() {
        switch (radioButton) {
            case ALL:
                breadthFirstSearch();
                break;
            case SINGLE:
                goalNode = singleChosenSquare;
                breadthFirstSearch();
                break;
            case PATH:
                goalNode = pathSquares.remove(0);
                breadthFirstSearch();
                break;
            default:
                System.out.println("PathPlannerWorkerError");
        }
        return null;
    }

    @Override
    protected void done() {
        if (!(radioButton == RadioButtons.SINGLE &&
            pathDirections.size() == 1)) {
            stopButton.setEnabled(true);
        }
        switch (radioButton) {
            case ALL:
                direction =
                    pathDirections.remove(pathDirections.size()-1);
                (directionWorker = new MoveWorker()).execute();
                break;
            case SINGLE:
                direction =
                    pathDirections.remove(pathDirections.size()-1);
                (directionWorker = new MoveWorker()).execute();
                break;
            case PATH:
                direction =
                    pathDirections.remove(pathDirections.size()-1);
                (directionWorker = new MoveWorker()).execute();
                break;
            default:
                System.out.println("PathPlannerWorkerError");
        }
        System.out.println("PathPlanner done");
    }
}

    private void
        abortButtonActionPerformed(java.awt.event.ActionEvent evt) {
        sketch.abortAction();
    }
```

```
private void
    leftButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (isReachable(presentPositionRow,
        presentPositionColumn-1)) {
        direction = Directions.LEFT;
        (directionWorker = new MoveWorker()).execute();
        directionButtonPressedEnabling(true);
    } else {
        System.out.println("KlimBot can't move left");
    }
}

private void
    downButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (isReachable(presentPositionRow+1,
        presentPositionColumn)) {
        direction = Directions.DOWN;
        (directionWorker = new MoveWorker()).execute();
        directionButtonPressedEnabling(true);
    } else {
        System.out.println("KlimBot can't move down");
    }
}

private void
    rightButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (isReachable(presentPositionRow,
        presentPositionColumn+1)) {
        direction = Directions.RIGHT;
        (directionWorker = new MoveWorker()).execute();
        directionButtonPressedEnabling(true);
    } else {
        System.out.println("KlimBot can't move right");
    }
}

private void
    startButtonActionPerformed(java.awt.event.ActionEvent evt) {
    startButton.setEnabled(false);
    if (radioButton == RadioButtons.ALL) {
        for (int i = 0; i < squareGrid.length; i++) {
            for (int j = 0; j < squareGrid[0].length; j++) {
                squareGrid[i][j].boltVisited = false;
            }
        }
        squareGrid[presentPositionRow][presentPositionColumn].boltVisited
        = true;
    }
    (pathPlannerWorker = new PathPlannerWorker()).execute();
}

private void
    stopButtonActionPerformed(java.awt.event.ActionEvent evt) {
    stopPressed = true;
```

```
        stopButton.setEnabled(false);
    }

    private void
        manualControlButtonActionPerformed(java.awt.event.ActionEvent
        evt) {
        startButton.setEnabled(false);
        stopButton.setEnabled(false);
        abortButton.setEnabled(false);
        if (isReachable(presentPositionRow-1,
            presentPositionColumn)) {
            upButton.setEnabled(true);
        }
        if (isReachable(presentPositionRow,
            presentPositionColumn-1)) {
            leftButton.setEnabled(true);
        }
        if (isReachable(presentPositionRow,
            presentPositionColumn+1)) {
            rightButton.setEnabled(true);
        }
        if (isReachable(presentPositionRow+1,
            presentPositionColumn)) {
            downButton.setEnabled(true);
        }
        radioButton = RadioButtons.MANUAL;
        cleanBolts();
    }

    private void
        singleBoltButtonActionPerformed(java.awt.event.ActionEvent
        evt) {
        enableAllButtons(false);
        radioButton = RadioButtons.SINGLE;
        cleanBolts();
    }

    private void
        choosePathButtonActionPerformed(java.awt.event.ActionEvent
        evt) {
        enableAllButtons(false);
        radioButton = RadioButtons.PATH;
        cleanBolts();
    }

    void directionButtonPressedEnabling(boolean start) {
        abortButton.setEnabled(start);
        if (!start && isReachable(presentPositionRow-1,
            presentPositionColumn)) {
            upButton.setEnabled(true);
        } else {
            upButton.setEnabled(false);
        }
        if (!start && isReachable(presentPositionRow,
            presentPositionColumn-1)) {
            leftButton.setEnabled(true);
        } else {
            leftButton.setEnabled(false);
        }
        if (!start && isReachable(presentPositionRow,
            presentPositionColumn+1)) {
            rightButton.setEnabled(true);
        } else {
            rightButton.setEnabled(false);
        }
        if (!start && isReachable(presentPositionRow+1,
            presentPositionColumn)) {
            downButton.setEnabled(true);
        } else {
            downButton.setEnabled(false);
        }
        manualControlButton.setEnabled(!start);
        allBoltsButton.setEnabled(!start);
        singleBoltButton.setEnabled(!start);
        choosePathButton.setEnabled(!start);
    }

    void enableAllButtons(boolean enable) {
        startButton.setEnabled(enable);
        stopButton.setEnabled(enable);
        abortButton.setEnabled(enable);
        upButton.setEnabled(enable);
        leftButton.setEnabled(enable);
        rightButton.setEnabled(enable);
        downButton.setEnabled(enable);
    }

    void breadthFirstSearch() {
        pathDirections.clear();
        Square node =
            squareGrid[presentPositionRow][presentPositionColumn];
        node.parent = null;
        nodeQueue.clear();
        while(!visitedNodes.isEmpty()) {
            visitedNodes.remove(0).nodeVisited = false;
        }
        node.nodeVisited = true;
        visitedNodes.add(node);
        nodeQueue.add(node);
        boolean goalNodeFound = false;
        while(!nodeQueue.isEmpty()) {
            node = nodeQueue.remove(0);
            if (radioButton == RadioButtons.ALL) {
                if (!node.boltVisited) {
                    node.boltVisited = true;
                    goalNodeFound = true;
                    break;
            }
        } else {
            if (node == goalNode) {
                goalNodeFound = true;
                break;
            }
        }
        for (int i = 0; i < 4; i++) {
            Square child = node.children[i];
            if (child != null && !child.nodeVisited) {
                child.parent = node;
                child.nodeVisited = true;
                visitedNodes.add(child);
                nodeQueue.add(child);
            }
        }
    }
    if (!goalNodeFound) {
        System.out.println("BFSError");
    } else {
        //backtracking
        while(node.parent != null) {
            if (node.parent.children[0] == node) {
                pathDirections.add(Directions.UP);
            } else if (node.parent.children[1] == node) {
                pathDirections.add(Directions.DOWN);
            } else if (node.parent.children[2] == node) {
                pathDirections.add(Directions.RIGHT);
            } else {
                pathDirections.add(Directions.LEFT);
            }
            node = node.parent;
        }
    }
}

void cleanBolts() {
    Square s = singleChosenSquare;
    if (s != null) {
        s.background = imgBolt;
        s.drawImage(s.forground, false);
        s = null;
    }
    while (!pathSquares.isEmpty()) {
        s = pathSquares.remove(0);
        s.background = imgBolt;
        s.drawImage(s.forground, false);
    }
    goalNode = null;
}

boolean isReachable(int row, int column) {
    if (row >= 0 && row < squareGrid.length && column >= 0 &&
     column < squareGrid[0].length) {
        return squareGrid[row][column].isReachable;
    }
    return false;
}

//assume both images are of the same size
ImageIcon combinedIcons(ImageIcon backgroundIcon, ImageIcon
    forgroundIcon) {
    if (backgroundIcon == null) {
        return forgroundIcon;
    } else if (forgroundIcon == null) {
        return backgroundIcon;
    }
    BufferedImage background =
        imageIconToBufferedImage(backgroundIcon);
    BufferedImage forground =
        imageIconToBufferedImage(forgroundIcon);
    BufferedImage combinedImage = new BufferedImage(
            background.getWidth(),
            background.getHeight(),
            BufferedImage.TYPE_INT_ARGB );
    Graphics2D g = combinedImage.createGraphics();
    g.drawImage(background,0,0,null);
    g.drawImage(forground,0,0,null);
    g.dispose();
    return new ImageIcon(combinedImage);
}

BufferedImage imageIconToBufferedImage(ImageIcon icon) {
    BufferedImage bufferedImage = new BufferedImage(
    icon.getIconWidth(),
    icon.getIconHeight(),
    BufferedImage.TYPE_INT_ARGB);
    Graphics g = bufferedImage.createGraphics();
    // paint the Icon to the BufferedImage.
    icon.paintIcon(null, g, 0,0);
    g.dispose();
    return bufferedImage;
}


public static void main(String args[]) {
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info :
        javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {
        java.util.logging.Logger.getLogger(KlimBotControl.class.getName())
        .log(java.util.logging.Level.SEVERE, null, ex);
    } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(KlimBotControl.class.getName())
            .log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(KlimBotControl.class.getName())
            .log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(KlimBotControl.class.getName())
            .log(java.util.logging.Level.SEVERE, null, ex);
        }

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new KlimBotControl().setVisible(true);
            }
        });
    }

    // Variables
    private javax.swing.JButton abortButton;
    private javax.swing.JRadioButton allBoltsButton;
    private javax.swing.JRadioButton choosePathButton;
    private javax.swing.JButton downButton;
    private javax.swing.JButton leftButton;
    private javax.swing.JRadioButton manualControlButton;
    private javax.swing.ButtonGroup pathChoiceGroup;
    private javax.swing.JButton rightButton;
    private javax.swing.JRadioButton singleBoltButton;
    private javax.swing.JPanel squareGridPanel;
    private javax.swing.JScrollPane squareGridScrollPane;
    private javax.swing.JButton startButton;
    private javax.swing.JButton stopButton;
    private javax.swing.JButton upButton;

   MoveWorker directionWorker;
      PathPlannerWorker pathPlannerWorker;

      int imageSize;
      String comNmb;
      Square[][] squareGrid;
      int presentPositionRow;
      int presentPositionColumn;
      int reachableSquaresCnt = -1; //not counting current position
      int visitedBoltsCnt = 0;
      ProcessingSketch sketch;
      enum Directions {UP, DOWN, RIGHT, LEFT}
      enum RadioButtons {MANUAL, ALL, SINGLE, PATH}
      Directions direction;
      RadioButtons radioButton = RadioButtons.MANUAL;
      boolean stopPressed = false;
      boolean isMoving = false;
      //for Breadth-First-Search
      Square singleChosenSquare, goalNode;
      ArrayList<Square> pathSquares = new ArrayList<>();
    ArrayList<Square> nodeQueue = new ArrayList<>();
    ArrayList<Square> visitedNodes = new ArrayList<>();
    //for storing the moves of a path
    ArrayList<Directions> pathDirections = new ArrayList<>();


 //ImageIcons for drawing bolts and KlimBot
 ImageIcon imgBolt = new javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/bolt.png"));
    ImageIcon imgUnreachableBolt = new
        javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/unreachableBolt.png"));
    ImageIcon imgChosenBolt = new javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/chosenBolt.png"));
    ImageIcon imgKlimBotBase = new javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/klimBoBase.png"));
    ImageIcon imgKlimBotLegUp = new
        javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/klimBoLegUp.png"));
    ImageIcon imgKlimBotLegLeft = new
        javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/klimBoLegLeft.png"));
    ImageIcon imgKlimBotLegRight = new
        javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/klimBoLegRight.png"));
    ImageIcon imgKlimBotLegDown = new
        javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/klimBoLegDown.png"));
    ImageIcon imgKlimBotThighUp = new
        javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/klimBoThighUp.png"));
    ImageIcon imgKlimBotThighLeft = new
        javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/klimBoThighLeft.png"));
    ImageIcon imgKlimBotThighRight = new
        javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/klimBoThighRight.png"));
    ImageIcon imgKlimBotThighDown = new
        javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/klimBoThighDown.png"));
    ImageIcon imgKlimBotOutstretchedInnerLegUp =
    new javax.swing.ImageIcon(getClass()
       .getResource("/KlimBotPack/klimBoOutstretchedInnerLegUp.png"));
    ImageIcon imgKlimBotOutstretchedInnerLegLeft =
    new javax.swing.ImageIcon(getClass()
       .getResource("/KlimBotPack/klimBoOutstretchedInnerLegLeft.png"));
    ImageIcon imgKlimBotOutstretchedInnerLegRight =
 new javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/klimBoOutstretchedInnerLegRight.png"));
    ImageIcon imgKlimBotOutstretchedInnerLegDown =
 new javax.swing.ImageIcon(getClass()
    .getResource("/KlimBotPack/klimBoOutstretchedInnerLegDown.png"));
    ImageIcon imgKlimBotOutstretchedOuterLegUp =
    new javax.swing.ImageIcon(getClass()
      .getResource("/KlimBotPack/klimBoOutstretchedOuterLegUp.png"));
   ImageIcon imgKlimBotOutstretchedOuterLegLeft =
new javax.swing.ImageIcon(getClass()
   .getResource("/KlimBotPack/klimBoOutstretchedOuterLegLeft.png"));
   ImageIcon imgKlimBotOutstretchedOuterLegRight =
   new javax.swing.ImageIcon(getClass()
      .getResource("/KlimBotPack/klimBoOutstretchedOuterLegRight.png"));
   ImageIcon imgKlimBotOutstretchedOuterLegDown =
   new javax.swing.ImageIcon(getClass()
      .getResource("/KlimBotPack/klimBoOutstretchedOuterLegDown.png"));
   ImageIcon imgKlimBotOutstretchedThighUp =
   new javax.swing.ImageIcon(getClass()
      .getResource("/KlimBotPack/klimBoOutstretchedThighUp.png"));
   ImageIcon imgKlimBotOutstretchedThighLeft =
   new javax.swing.ImageIcon(getClass()
      .getResource("/KlimBotPack/klimBoOutstretchedThighLeft.png"));
   ImageIcon imgKlimBotOutstretchedThighRight =
   new javax.swing.ImageIcon(getClass()
      .getResource("/KlimBotPack/klimBoOutstretchedThighRight.png"));
   ImageIcon imgKlimBotOutstretchedThighDown =
   new javax.swing.ImageIcon(getClass()
      .getResource("/KlimBotPack/klimBoOutstretchedThighDown.png"));


class Square extends javax.swing.JLabel {
    boolean hasBolt = true;
    boolean isReachable = false;
    ImageIcon background;
    ImageIcon forground;

    //for Breadth-First-Search
    Square[] children = new Square[4]; //up, down, right, left
    Square parent;
    boolean nodeVisited = false;

    //for Breadht-First-Search when visiting ALL bolts
    boolean boltVisited = false;

    Square() {
        setPreferredSize(new Dimension(20, 20));
        setOpaque(true);
        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                squareClicked();
            }
        });
    }

     void squareClicked() {
         if (!isMoving) {
         System.out.println("square clicked, isReachable: " +
             isReachable);
         // if singleBolt is selected
         if (radioButton == RadioButtons.SINGLE && isReachable &&
            squareGrid[presentPositionRow][presentPositionColumn] !=
                this) {
                startButton.setEnabled(true);
                if (singleChosenSquare != null) {
                    singleChosenSquare.background = imgBolt;
                    singleChosenSquare.setIcon(combinedIcons(singleChosenSquare
                  .background, singleChosenSquare.forground));
                }
                if (singleChosenSquare == this) {
                    singleChosenSquare = null;
                    startButton.setEnabled(false);
                } else {
                    singleChosenSquare = this;
                    background = imgChosenBolt;
                    setIcon(combinedIcons(background, forground));
                }
            }
            // if choosePath is selected
            if (radioButton == RadioButtons.PATH && isReachable) {
                if (!(pathSquares.isEmpty() &&
               squareGrid[presentPositionRow][presentPositionColumn]
                  == this)) {
                    startButton.setEnabled(true);
                    if (pathSquares.contains(this)) {
                        background = imgBolt;
                        pathSquares.remove(this);
                        if (pathSquares.isEmpty()) {
                            startButton.setEnabled(false);
                        } else if (pathSquares.size() == 1 &&
                     pathSquares.contains(squareGrid[presentPositionRow]
                        [presentPositionColumn])) {
                            Square s = pathSquares.remove(0);
                            s.background = imgBolt;
                            s.setIcon(combinedIcons(s.background,
                                s.forground));
                            startButton.setEnabled(false);
                        }
                    } else {
                        background = imgChosenBolt;
                        pathSquares.add(this);
                    }
                    setIcon(combinedIcons(background, forground));
                }
            }
            } else {
                System.out.println("KlimBot is moving, can't click
                    square.");
            }
        }

        void drawImage(ImageIcon forground, boolean hidesBackground)
            {
            this.forground = forground;
            if (hidesBackground) {
                setIcon(forground);
            } else {
                setIcon(combinedIcons(background, forground));
            }
        }

        void removeForground() {
            forground = null;
            setIcon(background);
        }
    }

  class ProcessingSketch extends PApplet {

        //variables
        Serial myPort;
        Servo servo;

        int servoCount = 8;
        int maxLoad = 1600;
        int normalSpeed = 150; //max relative ratio used is 1.4
        int angleLimitCCW = 750;
        int angleLimitCW = 200;

        boolean goodToGo = true;
        boolean[] servoIsAssignedMove = new boolean[servoCount + 1];
        boolean[] hasBeenStopped = new boolean[servoCount + 1];

        public void setup() {
            initRobot();
        }

        public void draw() {
        }

        void initRobot() {
            servo = new Servo();
            servo.init(this, comNmb, 1000000);

            //initializes speed and angleLimits
            for (int i = 1; i <= 8; i++) {
                servo.setMovingSpeed(i, normalSpeed);
                servo.setAngleLimitCCW(i, angleLimitCCW);
                servo.setAngleLimitCW(i, angleLimitCW);;
            }

            //initiate KlimBot's starting position
            setGoalPos(1, 641);
            setGoalPos(2, 249);
            setGoalPos(3, 578);
            setGoalPos(4, 282);
            setGoalPos(5, 641);
            setGoalPos(6, 249);
            setGoalPos(7, 648);
```

```
      setGoalPos(8, 251);
  }

void setGoalPos(int id, int pos) {
   if (goodToGo && !hasBeenStopped[id]) {
      servo.setGoalPosition(id, pos);
       servoIsAssignedMove[id] = true;
   }
}

  boolean servosAreMoving() {
   for (int i = 1; i <= servoCount; i++) {
      if (servoIsAssignedMove[i]) {
            if (servo.moving(i)) {
            return true;
            } else {
            servoIsAssignedMove[i] = false;
            }
      }
   }
return false;
}

//method for measuring the load of all servos
//disable servo whenever threshold 'maxValue' is exceeded
  void loadControl() {
   int tempLoad;
   while (servosAreMoving ()) {
      for (int i = 1; i <= servoCount; i++) {
            tempLoad = servo.presentLoad(i);
         if (tempLoad > maxLoad) {
            servo.setTorqueEnable(i, false);
            hasBeenStopped[i] = true;
            println("ALERT! Servo #" + i + " disabled, load (" +
               tempLoad + ") exceeding max load (" + maxLoad +
                   ")");
            abortAction();

            }
      }
   }
}

  void abortAction() {
   for (int i = 1; i <= servoCount; i++) {
      setGoalPos(i, servo.presentPosition(i));
   }
   goodToGo = false;
   println("ACTION ABORTED!");
}

  void setRelativeSpeed(int id, double ratio) {
   int newSpeed = (int) (ratio * normalSpeed);
   if (!hasBeenStopped[id]) {
        servo.setMovingSpeed(id, newSpeed);
    }
}

///////////////////// START - LEFT //////////////////////
  // KlimBot makes one step leftwards
  void moveLeft() {

   //lift up in order to release left leg
   setGoalPos(3, 627);
   setGoalPos(4, 288);
   setGoalPos(7, 656);
   setGoalPos(8, 228);
   loadControl(); //wait

   //release left leg
   setGoalPos(5, 664);
   setGoalPos(6, 256);
   loadControl(); //wait

   //stretch out left leg
   setRelativeSpeed(6, 1.14);
   setGoalPos(5, 409);
   setGoalPos(6, 546);
   loadControl(); //wait

   //draw KlimBot with left leg outstretched
   if (goodToGo) {
      drawKlimBotReachLeft();
   }

   //grips the bolt
   setRelativeSpeed(6, 0.5);
   setRelativeSpeed(5, 0.5);
   setGoalPos(5, 360);
   setGoalPos(6, 560);
   loadControl(); //wait

   //tightens grip
   setGoalPos(5, 394);
   setGoalPos(6, 537);
   loadControl(); //wait
   setRelativeSpeed(5, 1.0);
   setRelativeSpeed(6, 1.0);

   //lifts body for being able to release lower leg
   setGoalPos(7, 687);
   setGoalPos(8, 226);
   setGoalPos(3, 672);
   setGoalPos(4, 220);
   loadControl(); //wait

   //release lower leg
   setGoalPos(3, 739);
setGoalPos(4, 214);
loadControl(); //wait

//puts lower leg against wall for support
setGoalPos(3, 471);
setGoalPos(4, 388);
loadControl(); //wait

//release upper leg
setGoalPos(7, 630);
setGoalPos(8, 281);
loadControl(); //wait

setGoalPos(7, 672);
setGoalPos(8, 280);
loadControl(); //wait

//move KlimBot leftwards
setRelativeSpeed(5, 0.86);
setRelativeSpeed(1, 0.86);
setGoalPos(5, 649);
setGoalPos(6, 244);
setGoalPos(1, 392);
setGoalPos(2, 544);
loadControl(); //wait
setRelativeSpeed(1, 1.0);
setRelativeSpeed(5, 1.0);

//draw KlimBot moved left
if (goodToGo) {
   drawKlimBotDragLeft();
}

//grip bolt with upper leg
setGoalPos(7, 599);
setGoalPos(8, 286);
loadControl(); //wait

setGoalPos(7, 608);
setGoalPos(8, 272);
loadControl(); //wait

//trekker tilbake bakbeinet
//lift body for reattaching lower leg
setRelativeSpeed(4, 0.55);
setGoalPos(7, 685);
setGoalPos(8, 232);
setGoalPos(3, 693);
setGoalPos(4, 246);
loadControl(); //wait
setRelativeSpeed(4, 1.0);

//place lower leg over bolt
setGoalPos(3, 687);
```

```
    setGoalPos(4, 216);
    loadControl(); //wait

    //lift robot for releasing right leg
    setGoalPos(3, 627);
    setGoalPos(4, 288);
    setGoalPos(7, 656);
    setGoalPos(8, 228);
    setGoalPos(5, 641);
    setGoalPos(6, 249);
    loadControl(); //wait

    //release right leg
    setGoalPos(1, 391);
    setGoalPos(2, 571);
    loadControl(); //wait

    //retract right leg
    setRelativeSpeed(2, 1.14);
    setGoalPos(1, 668);
    setGoalPos(2, 254);
    loadControl(); //wait

    setRelativeSpeed(2, 0.22);
    setGoalPos(1, 610);
    setGoalPos(2, 267);
    loadControl(); //wait
    setRelativeSpeed(2, 1.0);

    //draw KlimBot after completing step
    if (goodToGo) {
          drawKlimBotRetractLeft();
    }

    //resume starting position
    setGoalPos(1, 641);
    setGoalPos(2, 249);
    setGoalPos(3, 578);
    setGoalPos(4, 282);
    setGoalPos(5, 641);
    setGoalPos(6, 249);
    setGoalPos(7, 648);
    setGoalPos(8, 251);
    loadControl(); //wait
}
  //////////////////// END - LEFT ///////////////////////////

  //////////////////////// START - RIGHT ////////////////////
  // KlimBot makes one step rightwards
  void moveRight() {

   //lift up for releasing right leg
   setGoalPos(3, 627);
   setGoalPos(4, 288);
setGoalPos(7, 656);
setGoalPos(8, 228);
loadControl(); //wait

//release right leg
setGoalPos(1, 664);
setGoalPos(2, 256);
loadControl(); //wait

//stretch out right leg
setRelativeSpeed(2, 1.14);
setGoalPos(1, 409);
setGoalPos(2, 546);
loadControl(); //wait

//draw KlimBot with right leg outstretched
if (goodToGo) {
   drawKlimBotReachRight();
}

//grip bolt
setRelativeSpeed(2, 0.5);
setRelativeSpeed(1, 0.5);
setGoalPos(1, 360);
setGoalPos(2, 560);
loadControl(); //wait

//tighten grip
setGoalPos(1, 394);
setGoalPos(2, 537);
loadControl(); //wait
setRelativeSpeed(1, 1.0);
setRelativeSpeed(2, 1.0);

//lift body for releasing lower leg
setGoalPos(7, 687);
setGoalPos(8, 226);
setGoalPos(3, 672);
setGoalPos(4, 220);
loadControl(); //wait

//release lower leg
setGoalPos(3, 739);
 setGoalPos(4, 214);
 loadControl(); //wait

 //place lower leg against wall for support
 setGoalPos(3, 471);
 setGoalPos(4, 388);
 loadControl(); //wait

 //release upper leg
setGoalPos(7, 630);
setGoalPos(8, 281);
```

```
loadControl(); //wait

setGoalPos(7, 672);
setGoalPos(8, 280);
loadControl(); //wait

//move KlimBot rightwards
setRelativeSpeed(1, 0.86);
setRelativeSpeed(5, 0.86);
setGoalPos(1, 649);
setGoalPos(2, 244);
setGoalPos(5, 392);
setGoalPos(6, 544);
loadControl(); //wait
setRelativeSpeed(1, 1.0);
setRelativeSpeed(5, 1.0);

//draw KlimBot moved to the right
if (goodToGo) {
   drawKlimBotDragRight();
}

//grip bolt with upper leg
setGoalPos(7, 599);
setGoalPos(8, 286);
loadControl(); //wait

setGoalPos(7, 608);
setGoalPos(8, 272);
loadControl(); //wait

//lift body for reattaching lower leg
setRelativeSpeed(4, 0.55);
setGoalPos(7, 685);
 setGoalPos(8, 232);
 setGoalPos(3, 693);
 setGoalPos(4, 246);
 loadControl(); //wait
 setRelativeSpeed(4, 1.0);

//place lower leg over bolt
setGoalPos(3, 687);
 setGoalPos(4, 216);
 loadControl(); //wait

 //lift body for releasing left leg
 setGoalPos(3, 627);
  setGoalPos(4, 288);
 setGoalPos(7, 656);
 setGoalPos(8, 228);
 setGoalPos(1, 641);
 setGoalPos(2, 249);
 loadControl(); //wait
```

```
        //release left leg
        setGoalPos(5, 391);
        setGoalPos(6, 571);
        loadControl(); //wait

        //retract left leg
        setRelativeSpeed(6, 1.14);
        setGoalPos(5, 668);
        setGoalPos(6, 254);
        loadControl(); //wait

        setRelativeSpeed(6, 0.22);
        setGoalPos(5, 610);
        setGoalPos(6, 267);
        loadControl(); //wait
        setRelativeSpeed(6, 1.0);

        //draw KlimBot after move is complete
        if (goodToGo) {
           drawKlimBotRetractRight();
        }

        //resume starting position
        setGoalPos(1, 641);
        setGoalPos(2, 249);
        setGoalPos(3, 578);
        setGoalPos(4, 282);
        setGoalPos(5, 641);
        setGoalPos(6, 249);
        setGoalPos(7, 648);
        setGoalPos(8, 251);
        loadControl(); //wait
    }
      /////////////////////// END - RIGHT ///////////////////////

      ////////////////////// START - UP //////////////////////
      // KlimBot makes one step upwards
      void moveUp() {

       //release upper leg
        setGoalPos(7, 650);
        setGoalPos(8, 283);
       loadControl(); //wait

        //stretches out upper leg
        setRelativeSpeed(7, 0.83);
        setGoalPos(7, 382);
        setGoalPos(8, 595);
        loadControl(); //wait

        //draw KlimBot with uoper leg outstretched
        if (goodToGo) {
            drawKlimBotReachUp();
        }
```

```
//grip bolt
setRelativeSpeed(7, 0.5);
setRelativeSpeed(8, 0.5);
setGoalPos(7, 360);
setGoalPos(8, 551);
loadControl(); //wait

//tighten grip
setGoalPos(7, 382);
loadControl(); //wait
setRelativeSpeed(7, 1.0);
setRelativeSpeed(8, 1.0);

//lift body for releasing side grippers
setGoalPos(3, 550);
setGoalPos(4, 320);
setGoalPos(7, 420);
setGoalPos(8, 510);
loadControl(); //wait

//release side grippers
setGoalPos(1, 650);
setGoalPos(2, 300);
setGoalPos(5, 650);
setGoalPos(6, 300);
loadControl(); //wait

//move KlimBot upwards
setRelativeSpeed(3, 0.85);
setRelativeSpeed(4, 1.09);
setRelativeSpeed(8, 1.09);
setGoalPos(7, 705);
setGoalPos(8, 218);
setGoalPos(3, 308);
setGoalPos(4, 646);
loadControl(); //wait
setRelativeSpeed(3, 1.0);
setRelativeSpeed(4, 1.0);
setRelativeSpeed(8, 1.0);

//draw KlimBot after moving upwards
if (goodToGo) {
   drawKlimBotDragUp();
}

//reattach side grippers
setRelativeSpeed(1, 0.5);
setRelativeSpeed(2, 0.5);
setRelativeSpeed(5, 0.5);
setRelativeSpeed(6, 0.5);
setGoalPos(1, 597);
setGoalPos(2, 273);
setGoalPos(5, 597);
```

```
        setGoalPos(6, 273);
        loadControl(); //wait

       //tighten grip
        setGoalPos(1, 641);
        setGoalPos(2, 249);
        setGoalPos(5, 641);
        setGoalPos(6, 249);
        loadControl(); //wait
        setRelativeSpeed(1, 1.0);
        setRelativeSpeed(2, 1.0);
        setRelativeSpeed(5, 1.0);
        setRelativeSpeed(6, 1.0);

        //lift lower leg for release
        setGoalPos(3, 401);
        setGoalPos(4, 527);
        loadControl(); //wait

        //release lower leg
        setRelativeSpeed(3, 0.5);
        setGoalPos(3, 439);
        setGoalPos(4, 521);
        loadControl(); //wait
        setRelativeSpeed(3, 1.0);

        //retract lower leg
        setGoalPos(3, 713);
        setGoalPos(4, 214);
        loadControl(); //wait

        //draw KlimBot after move is complete
        if (goodToGo) {
            drawKlimBotRetractUp();
        }

       //resume starting position
        setGoalPos(3, 691);
        setGoalPos(4, 221);
        loadControl(); //wait
        setGoalPos(3, 578);
        setGoalPos(4, 282);
        setGoalPos(7, 648);
        setGoalPos(8, 251);
        loadControl(); //wait
    }
      ////////////////////// END - UP /////////////////////////

      ////////////////////// START - DOWN ////////////////////
      // KlimBot makes one step downwards
      void moveDown() {

       //lift body for releasing lower leg
        setGoalPos(7, 685);
```

```
setGoalPos(8, 232);
setGoalPos(3, 672);
setGoalPos(4, 220);
loadControl(); //wait

//release lower leg
setGoalPos(3, 739);
setGoalPos(4, 216);
loadControl(); //wait

//stretch out lower leg
setGoalPos(7, 705);
setGoalPos(8, 218);
setGoalPos(3, 429);
setGoalPos(4, 508);
loadControl(); //wait

//draw KlimBot with lower leg outstretched
if (goodToGo) {
    drawKlimBotReachDown();
}

//grip bolt with lower leg
setRelativeSpeed(3, 0.5);
setRelativeSpeed(4, 0.5);
setGoalPos(3, 370);
setGoalPos(4, 548);
loadControl(); //wait
setRelativeSpeed(3, 1.0);
setRelativeSpeed(4, 1.0);

//lift body for releasing side grippers
setGoalPos(3, 308);
setGoalPos(4, 646);
loadControl(); //wait

//release side grippers
setGoalPos(1, 650);
setGoalPos(2, 300);
setGoalPos(5, 650);
setGoalPos(6, 300);
loadControl(); //wait

//move KlimBot downwards
setRelativeSpeed(3, 0.94);
setRelativeSpeed(4, 1.40);
setRelativeSpeed(7, 1.10);
setRelativeSpeed(8, 1.02);
setGoalPos(3, 577);
setGoalPos(4, 301);
setGoalPos(7, 415);
setGoalPos(8, 511);
loadControl(); //wait
setRelativeSpeed(3, 1.0);
```

```
        setRelativeSpeed(4, 1.0);
        setRelativeSpeed(7, 1.0);
        setRelativeSpeed(8, 1.0);

        //draw KlimBot after moving down
        if (goodToGo) {
            drawKlimBotDragDown();
        }

        //reattach side grippers
        setRelativeSpeed(1, 0.5);
        setRelativeSpeed(2, 0.5);
        setRelativeSpeed(5, 0.5);
        setRelativeSpeed(6, 0.5);
        setGoalPos(1, 614);
        setGoalPos(2, 260);
        setGoalPos(5, 614);
        setGoalPos(6, 260);
        loadControl(); //wait
        setRelativeSpeed(1, 1.0);
        setRelativeSpeed(2, 1.0);
        setRelativeSpeed(5, 1.0);
        setRelativeSpeed(6, 1.0);

       //tighten grip
        setGoalPos(1, 641);
        setGoalPos(2, 249);
        setGoalPos(3, 578);
         setGoalPos(4, 282);
         setGoalPos(5, 641);
        setGoalPos(6, 249);
        setGoalPos(7, 376);
        setGoalPos(8, 586);
        loadControl(); //wait

        //retract upper leg
        setRelativeSpeed(8, 1.19);
        setGoalPos(7, 625);
        setGoalPos(8, 289);
        loadControl(); //wait

        //draw KlimBot when move is complete
        if (goodToGo) {
            drawKlimBotRetractDown();
        }

        //grip with upper leg
        setRelativeSpeed(7, 0.70);
        setRelativeSpeed(8, 0.5);
        setGoalPos(7, 605);
        setGoalPos(8, 259);
        loadControl(); //wait

        //tighten grip
```

```
          setGoalPos(7, 648);
          setGoalPos(8, 251);
          loadControl(); //wait
          setRelativeSpeed(7, 1.0);
          setRelativeSpeed(8, 1.0);
      }
        ////////////////////////// END - DOWN //////////////////////
    }
}
```